UNIVERSIDAD CARLOS III DE MADRID

Escuela Politécnica Superior

Departamento de Ingeniería de Sistemas y Automática

PROYECTO FIN DE CARRERA

INGENIERÍA INDUSTRIAL

Especialidad en Automática y Electrónica Industrial

"DETECCIÓN Y RECONOCIMIENTO DE SEMÁFOROS POR VISIÓN ARTIFICIAL"

Víctor Alonso Mendieta

2013

Título: Detección y Reconocimiento de Semáforos por Visión Artificial

Autor: Víctor Alonso Mendieta

Tutor: Daniel Olmeda Reino

Director: Arturo de la Escalera Hueso

Tribunal:

Presidente: José María Armingol Moreno

Vocal: David Martín Gómez

Secretario: Agapito Ledezma Espino

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 31 de Octubre de 2013 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de _____.

Firmas:

Presidente: José María Armingol Moreno

Vocal: David Martín Gómez

Secretario: Agapito Ledezma Espino

Agradecimientos

Una vez más, quisiera reservar estas primeras palabras para dar las gracias a mi madre, mis hermanos y mi abuela, por el cariño y el apoyo recibidos durante estos años de carrera. Y no me olvido de mi padre: que sepa que allá donde esté, en especial este proyecto se lo dedico a él.

Resumen

Este proyecto fin de carrera presenta un *Sistema Avanzado de Asistencia a la Conducción (ADAS)* que consiste en una aplicación software basada en visión artificial diseñada para detectar y reconocer semáforos a través de una cámara instalada a bordo de un vehículo. La principal característica que presenta esta aplicación es que utiliza un clasificadores en cascada junto con técnicas de extracción de características como *Haar-Like Features* y *Local Binary Patterns* para detectar a los semáforos. La aplicación ha sido programada en código *C++* haciendo uso de las funciones que proporcionan las librerías de *OpenCV* para el diseño de aplicaciones de visión artificial.

El clasificador de semáforos se ha obtenido mediante un proceso de entrenamiento con un conjunto de muestras de semáforos. Estas muestras han sido recopiladas a partir de grabaciones del tráfico tomadas desde una cámara móvil acoplada a un vehículo. Con el objetivo de ubicar las muestras en las grabaciones para su posterior extracción, se ha diseñado una aplicación en *Visual Basic* que dispone de un interfaz de usuario con el que se permite marcar sobre las imágenes grabadas la localización y el tamaño de las ventanas que contienen a los semáforos y guardar estas referencias en un archivo de etiquetado.

El entrenamiento del clasificador se realiza haciendo uso de las aplicaciones proporcionadas por las librerías de *OpenCV*. Adicionalmente, se han programado en *C++* otras aplicaciones con las que completar el entrenamiento, entre las que se encuentra una para preprocesar las imágenes de muestras y otra para evaluar el comportamiento del clasificador. Para gestionar todo este proceso de entrenamiento, se ha creado un modelo de directorio que contiene de forma organizada en carpetas, todos los archivos y aplicaciones necesarias para dicho entrenamiento. Con este modelo de organización se permite realizar la configuración y la ejecución de estas aplicaciones mediante unos simples archivos *script* programados en *Bash*.

Índice General

Capítulo 1

Introducción

1.1. Antecedentes

En la actualidad, el automóvil es el medio de transporte más utilizado por la población. Esto se debe, entre otras cosas, a la comodidad, autonomía y libertad que ofrece a sus usuarios frente a otros medios de transporte. Pero ligado al automóvil existen una serie de problemas que los usuarios deben considerar como el tráfico, la contaminación o los accidentes. El incremento del número de automóviles en los últimos años ha provocado que tanto los organismo de tráfico como los fabricantes pongan cada vez más atención en buscar soluciones para resolver estos problemas. Es un hecho que cada vez los automóviles contaminan menos, son más seguros y más inteligentes. El dotar de esa inteligencia a los automóviles ha sido una de esas líneas que más se han venido reforzando sobre todo gracias al gran desarrollo de la tecnología electrónica y de computación. Al fin y al cabo, la mayoría de los accidentes de tráfico son producto de errores humanos, por lo que si se elimina a este factor creando vehículos tan inteligentes que sean capaces de circular de forma autónoma a la vez que segura, el problema se habrá resuelto. Hasta el momento, el ejemplo más avanzado que existe de vehículo autónomo, es el desarrollado por la empresa *Google* en su proyecto *Google Driverless Car*.

Figura 1.1: *Google Driverless Car.*

Pero la verdad es que aunque la tecnología ha avanzado mucho, todavía está lejos de poder implementar un sistema que permita hacer que los vehículos circulen realmente de forma autónoma. Por esta razón, ahora mismo el desarrollo de vehículos inteligentes de forma comercial está centrado en una tecnología más cercana como son los **Sistemas Avanzados de Asistencia a la Conducción *(Advanced Driver Assistance Systems) (ADAS)* [1]**. Estos sistemas están basados en sensores de entorno como ultrasonidos, radares o cámaras de vídeo que detectan las inmediaciones del vehículo ayudando al conductor a hacer simplemente la conducción más cómoda o en situaciones críticas en las que es necesario actuar de forma rápida y segura. Entre los sistemas *ADAS* que actualmente existen, se encuentran:

- **Sistema de navegación a bordo *(In-vehicle navigation system)*.** Con *GPS* y *TMC* para ofrecer posicionamiento en carretera e información del tráfico.

- **Control de crucero adaptativo *(Adaptative cruise control)*.** Adapta automáticamente la velocidad del vehículo para mantener una distancia seguridad con los vehículos que le anteceden.

Figura 1.2: *Navegador de a bordo.*

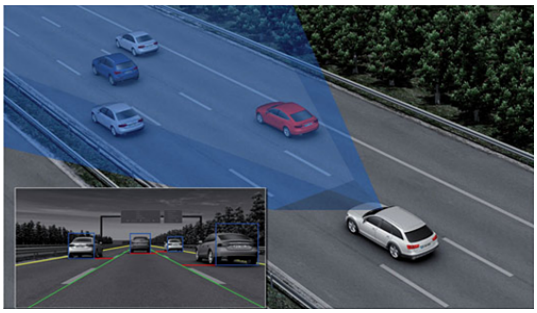


Figura 1.3: *Control de Crucero adaptativo.*

- **Asistencia de cambio de carril *(Lane change assistance)*.** Son sistemas que cubren el ángulo muerto de los vehículos avisando de la presencia de otros en sus inmediaciones.

- **Sistema de advertencia de abandono de carril *(Lane departure warning system)*.** Advierte al conductor cuando el vehículo comienza a salirse de su carril en autopistas y carreteras principales.

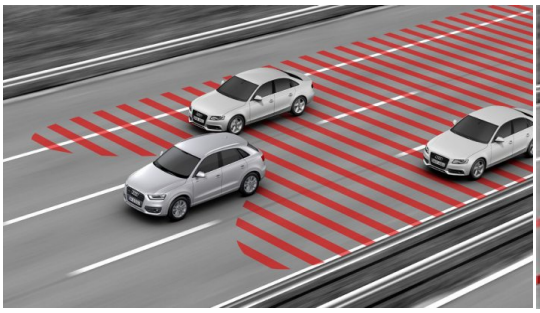


Figura 1.4: *Asistencia de cambio de carril.*

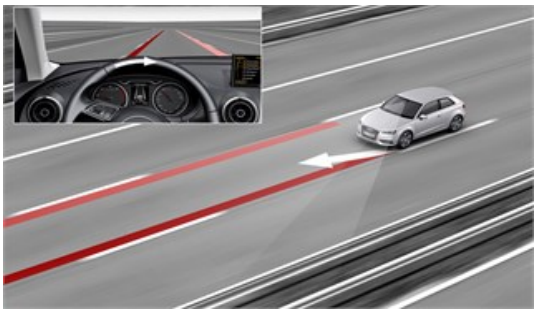


Figura 1.5: *Advertencia de abandono de carril.*

- **Sistema anticolisión *(Collision avoidance system)*.** Detecta obstáculos u otros vehículos detenidos en la vía y actúa sobre el freno del vehículo en caso de prever colisión.

- **Adaptación de velocidad inteligente *(Intelligent speed adaptation)*.** Este sistema monitoriza la velocidad límite de la carretera y advierte al conductor o actúa sobre el vehículo en caso de que este sobrepase dicho límite.

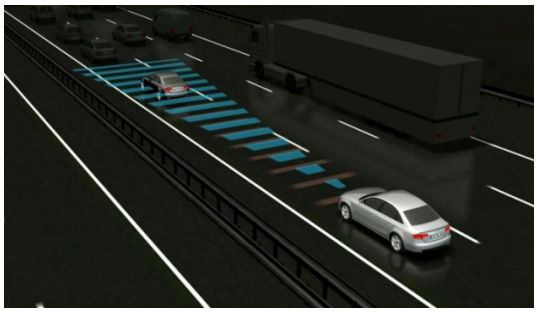


Figura 1.6: *Sistema anticolisión.*

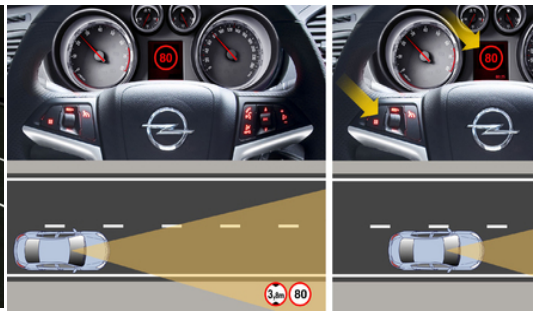


Figura 1.7: *Adaptación de velocidad inteligente.*

- **Reconocimiento de señales de tráfico *(Traffic sign recognition)*.** Capta las señales de tráfico y advierte al conductor sobre su presencia en la carretera.

- **Ayuda en aparcamiento *(Park assist)*.** Es un sistema que cuando se activa indica al conductor mediante alertas la cercanía a los obstáculos en la maniobra de aparcamiento.

Figura 1.8: *Reconocimiento de señales.*

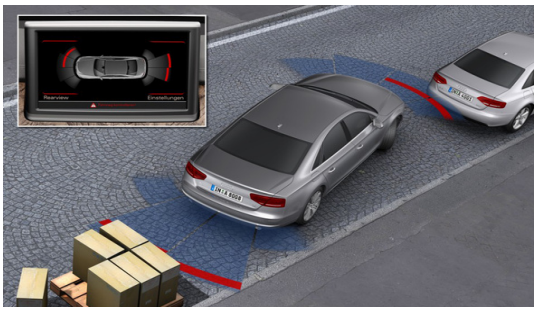


Figura 1.9: *Ayuda en aparcamiento.*

- **Control de luces adaptativo *(Adaptive light control)*.** Actúa sobre las luces de del vehículo en respuesta a las condiciones de visibilidad, dirección, suspensión, velocidad del vehículo, o presencia de otros vehículos.

- **Visión nocturna *(Automotive night visión)*.** Dispone de una pantalla donde se monitoriza la carretera en visión nocturna. Sirve para mejorar la percepción del conductor en la oscuridad.

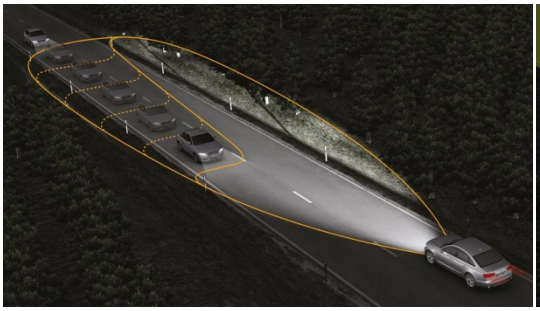


Figura 1.10: *Control de luces adaptativo.*

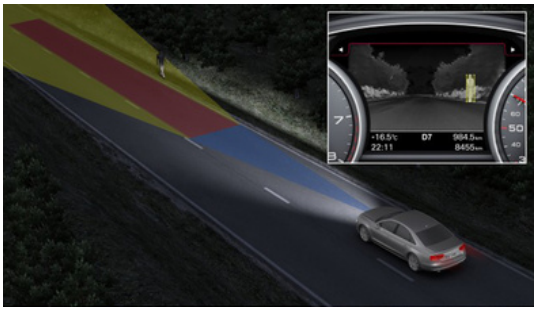


Figura 1.11: *Visión nocturna.*

- **Control de descenso *(Hill descend control)*.** Permite un descenso de pendientes suave y controlado en terrenos irregulares sin que el conductor tenga que pisar el freno.

- **Detección de somnolencia en el conductor *(Driver drowsiness detection)*.** Dispone de una cámara que monitoriza la cara del conductor y detecta cuando éste está somnoliento.

- **Sistemas de comunicación Vehicular *(Vehicular communication systems)*.** Son nodos que se sitúan en las carreteras y que se comunican con los vehículos transmitiéndoles información sobre el tráfico o advertencias de seguridad.

En definitiva, cada vez van surgiendo nuevos tipos de sistemas *ADAS* que van tomando más protagonismo en el proceso de conducción en busca de esa meta que es la conducción segura.

La aplicación que se presenta en este proyecto fin de carrera se incluye dentro de los sistemas *ADAS* destinados al reconocimiento de señales de tráfico. En concreto la aplicación está diseñada para detectar y reconocer los semáforos presentes en la carretera. El vehículo deberá disponer una cámara con la que se monitorice la carretera y un equipo que procese la aplicación. Mediante una pantalla se advertiría al conductor la presencia de los semáforos detectados y su correspondiente señalización.

Figura 1.12: *Semáforo.*

1.2. Objetivos

1.2.1. Objetivo Principal

De entre todos los objetivos que se han pretendido alcanzar con el desarrollo de este proyecto, el principal objetivo ha sido el de crear una aplicación software que permita detectar y reconocer semáforos mediante visión por computador a través de una cámara instalada a bordo de un vehículo. Esta aplicación no se ha planteado para su uso en un sistema que permita de forma autónoma controlar las maniobras del vehículo en presencia de un semáforo, sino que lo que se pretende es que sirva como sistema de apoyo al conductor en la conducción.

Dado que se está tratando con temas de seguridad vial, no solo se busca que la aplicación detecte y reconozca semáforos de cualquier manera. Es muy importante que la aplicación sea eficaz a la vez que segura. Con esto lo que se pretende decir es que aunque la aplicación no detecte y reconozca todos los semáforos que se capturen a través de la cámara, para aquellos que si lo haga, por pocos que sean, deberá hacerlo correctamente. Conseguido este requisito, la calidad de la aplicación vendrá dada por la cantidad de semáforos que es capaz de encontrar. En este sentido se buscará que sea lo mayor posible. Otro rasgo que debe tener la aplicación para que sea segura es la velocidad. Debe trabajar en tiempo real para que la comunicación con el conductor sea coordinada y no se produzcan errores de entendimiento. Esta es razón por la cual además de un sistema eficaz se necesita que sea rápido en respuesta.

La aplicación software que se diseñe será implementada en código *C++* haciendo uso de las funciones proporcionadas por la biblioteca libre *OpenCV* para el diseño de aplicaciones basadas en visión artificial.

1.2.2. Objetivos Secundarios

Si bien el objetivo principal es crear una aplicación software para la detección y reconocimiento de semáforos, existían una serie de trabajos previos que había que desarrollar y otros que han ido surgiendo, para alcanzar dicho objetivo. Estos trabajos han derivado en nuevos objetivos a los que se llamará secundarios, y son los siguientes:

- **Diseñar una aplicación software para el etiquetado de semáforos en imágenes.**

  El algoritmo utilizado para detectar semáforos, llamado clasificador, necesita ser entrenado con muestras de semáforos. Estas muestras son extraídas de imágenes a través de un archivo de etiquetado que indica la localización y el tamaño de los semáforos. Para que los resultados sean buenos es imprescindible tener una gran cantidad de muestras por lo que el trabajo de etiquetar semáforos en imágenes puede ser muy costoso en tiempo.

  Para reducir el tiempo de etiquetado, se pretende diseñar una aplicación software con un interfaz de usuario que permita, de forma cómoda y rápida, marcar los semáforos sobre las imágenes y generar automáticamente el archivo de etiquetado.

- **Crear una aplicación software para preprocesar las imágenes de muestra.**

  En las imágenes recolectadas para poder realizar el entrenamiento del detector, puede que los semáforos no se destaquen bien por motivos de mala iluminación, poco contraste o la presencia de ruido durante la toma de las imágenes. Este inconveniente puede ser la causa de que el rendimiento de detector, una vez entrenado, no sea todo lo bueno que se esperaba, ya que el algoritmo de entrenamiento no habrá podido extraer bien los rasgos de los semáforos.

  Para solucionar este problema, se tratará de diseñar una aplicación que permita preprocesar las imágenes de muestra con el fin de mejorar su calidad. Esta aplicación será configurable mediante una serie de parámetros que permitan seleccionar el tipo de preprocesamiento que mejor se ajuste a las condiciones que presenten las imágenes.

- **Desarrollar una aplicación software para evaluar el rendimiento de los clasificadores.**

  Como ya se ha comentado antes, un clasificador será el encargado de detectar los semáforos, y éste es creado a partir del entrenamiento con varias muestras de ellos. Para poder conocer las características del clasificador será necesario someterlo a un test.

  Con esta idea, se pretende crear una aplicación que calcule indicadores y gráficas de rendimiento para poder evaluar los resultados del entrenamiento y valorar la calidad del clasificador. Estos indicadores y gráficas se utilizarán para ajustar parámetros del entrenamiento, comparar clasificadores y sobre todo medir la eficacia del clasificador.

- **Crear un modelo de organización de carpetas, archivos y aplicaciones, para el proceso de creación de muestras, entrenamiento y test de clasificadores.**

  El proceso de creación de muestras, entrenamiento y test de clasificadores, es una tarea en la que intervienen muchos elementos (carpetas, archivos y aplicaciones) que guardan relación a través de direccionamientos entre sí. Esta cantidad de elementos es difícil de manejar si no se tiene una gestión de carpetas ordenada.

  Para facilitar esta tarea, se pretende crear un modelo de organización de carpetas, archivos y aplicaciones, controlado por archivos scripts. Los scripts contendrán los parámetros de configuración y comandos necesarios para hacer comunicar y coordinar entre sí a los distintos elementos. Así pues, con pocos clics y teclado, se podrán compilar y construir aplicaciones, configurar los parámetros de las aplicaciones y ejecutarlas, crear informes con los resultados de ejecución, definir la ubicación de almacenamiento de archivos, etc.

Capítulo 2

Fundamentos Teóricos

2.1. Características

2.1.1. Introducción

Todas las técnicas de detección de objetos con visión por computador se basan en una misma idea aparentemente sencilla de entender pero a la vez difícil de encontrar. Esta idea consiste en identificar aquella característica o conjunto de características del objeto a detectar que permitan distinguirle del entorno que lo rodea en una imagen.

Algunas de las características que distingan a un objeto podrían ser el color, el tamaño o la forma, pero esto no suele ser suficiente. En la mayoría de los casos varios objetos pueden compartir estas mismas características o incluso que estas características varíen para un mismo objeto en función de las condiciones en las que se perciban (distancia, perspectiva, iluminación).

Esta no uniformidad en las características de un mismo objeto es la razón por la que su extracción basada en métodos clásicos, como la umbralización o la detección de bordes, no es suficiente dependiendo de para qué aplicaciones. Por este motivo, con el tiempo se han ido desarrollando un conjunto de técnicas de extracción de características más complejas que permiten adaptarse de alguna forma a este inconveniente.

Así pues, este apartado tiene como objetivo la descripción de dos de las técnicas modernas que han sido utilizadas para identificar las características de los objetos de estudio del proyecto, los semáforos. Más en concreto, se explicarán dos de las técnicas implementadas por las librerías de *OpenCV* para la detección de objetos, como son las **características *Haar*** y las **características *LBP***.

2.1.2. Características *Haar*

2.1.2.1. Definición

La primera versión de esta técnica fueron las llamadas *Haar wavelets* propuestas por Papageorgiou et al. en **[2]** para la detección de caras y peatones, pero su uso no fue muy significativo hasta unos años más tarde. Entonces Viola y Jones **[3]** adaptaron la idea desarrollando una nueva versión a la que denominaron **características *Haar (Haar-Like Features)***.

La idea que se propone esta técnica consiste en identificar aquellas rasgos que van a definir a un objeto en base a la estructura de los niveles de intensidad que presentan sus pixeles en una imagen. Esta información será pues extraída aplicando sobre la imagen del objeto una series de funciones implementadas bajo las denominadas características *Haar*. La ventaja de esta técnica es que permite detectar la estructura de los objetos aunque esta no sea uniforme.

Una característica *Haar* se pude definir como una ventana de pixeles, de tamaño y orientación variables, dividida en regiones rectangulares, pudiendo ser cada una de estas regiones de dos tipos a las que se les llamará positivas o negativas (en **[3]** se les llama blancas o grises). En la siguiente imagen se muestra el aspecto que puede presentar una característica *Haar* de cuatro pixeles:

Figura 2.1: *Característica Haar.*

Esta ventana de pixeles se va desplazando sobre la imagen de detección evaluando, por una parte, la suma de los pixeles que se caen sobre las regiones positivas y, por otra parte, la suma de los pixeles que caen sobre las regiones negativas. A la diferencia entre la suma de las regiones positivas y la suma de las regiones negativas será lo que se denomine el valor de la característica. Así pues, el valor de una característica *Haar* en un punto $H(x,y)$, se puede representar mediante la siguiente fórmula:

$$H(x,y) = \sum_{p} I(x,y) - \sum_{n} I(x,y) \tag{2.1}$$

donde $I(x,y)$ representa la imagen a evaluar, y p y n representan respectivamente las regiones positivas y negativas de la característica *Haar* sobre la imagen. Mediante esta operación se obtendrá un mapa con los valores de la característica en cada posición de la imagen de detección. Estos valores son utilizados después junto con un umbral para clasificar las diferentes regiones de la imagen.

Figura 2.2: *Proceso de aplicación de una característica Haar a una imagen. (a) Desplazamiento de la característica sobre la imagen. (b) Cálculo de los valores de la característica. (c) Umbralizado de los valores de la característica.*

2.1.2.2. Tipos de Características *Haar*

Una característica *Haar* va a estar definida por parámetros como el tamaño, la orientación o la distribución de las regiones positivas y negativas, pudiéndose construir infinidad de tipos. A su vez, estos parámetros van a depender de los rasgos del objeto a detectar, y más en concreto de la distribución de intensidades de los pixeles que componen dicho rasgo. Así pues, el objetivo a la hora de construir una característica *Haar* es buscar que su estructura se asemeje a la del rasgo a detectar. Según esto, se pueden encontrar características *Haar* para la detección de bordes, líneas, contornos, etc.

El set de características *Haar* propuesto por Viola y Jones en **[3]** es el mostrado en la ***Figura 2.3***. Se compone de tres tipos de características clasificadas según el número de regiones rectangulares de que disponen: de dos regiones, de tres regiones y de cuatro regiones. Las características de dos regiones se utilizan para detectar bordes, según la orientación, verticales (0°) u horizontales (90°). Las características de tres regiones detectan líneas verticales (0°) y horizontales (90°). Y por último las características de cuatro regiones detectan líneas diagonales.

Figura 2.3: *Características Haar utilizadas por Viola y Jones. (a) Característica de dos regiones. (b) Características tres regiones. (c) Característica de cuatro regiones.*

Lienhart y Maydt en su trabajo **[4]** fueron los siguientes en desarrollar una mejora para las características *Haar*. En este caso introdujeron el concepto de características *Haar* inclinadas (45°). Esto permitió aumentar la dimensión del set de características (ver ***Figura 2.4***) en un intento de mejorar la detección de objetos en imágenes. Así pues, con el nuevo set se consiguen detectar además bordes y líneas a 45°.

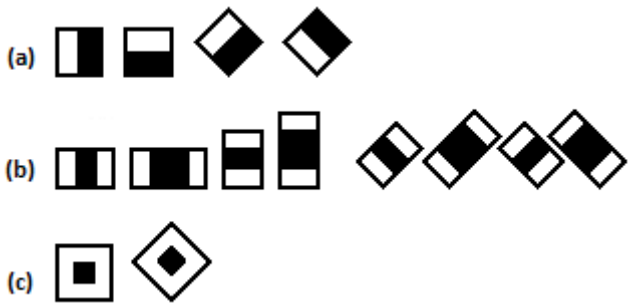


Figura 2.4: *Características Haar utilizadas por Lienhart y Maydt. (a) Características de bordes. (b) Características de líneas. (c) Características de centros-contornos.*

2.1.2.3. Ejemplo

Para entender mejor la idea de funcionamiento de las características *Haar*, se pondrá un ejemplo. Si se tuviese que detectar la localización de los ojos en una cara, se puede notar que los ojos presentan unos niveles de intensidad más oscuros que la parte de la nariz que los separa. Así pues, esta distribución de niveles de intensidad pude ser identificada por una característica *Haar* compuesta por dos regiones negativas unidas por una region positiva (ver ***Figura 2.5***).

Figura 2.5: *Detección de ojos en una imagen con una característica Haar.*

En la ***Figura 2.6*** se muestra de forma simplificada la carácterística *Haar* elegida junto con la zona de la imagen de los ojos donde es aplicada. Como se puede comprobar, el resultado obtenido es un valor elevado, es más, es el máximo valor que se puede dar en cualquier caso para esta característica. Esto es un indicativo de que es muy probable que los ojos se encuentren en esta posición.

$h(x,y)$

-1	-1	+1	+1	-1	-1
-1	-1	+1	+1	-1	-1

$I(x,y)$

0	0	255	255	0	0
0	0	255	255	0	0

$$H(x,y) = 2040$$

Figura 2.6: *Ejemplo de una característica Haar para detectar ojos aplicada sobre la región de la imagen de los ojos.*

Ahora se va comprobar el efecto de la misma característica al aplicarla sobre otra región de la imagen, como por ejemplo, sobre la frente que es una región uniforme de intensidades claras. En la ***Figura 2.7*** se puede ver de forma simplificada el resultado. En este caso se comprueba que el valor de la característica está lejos del máximo valor arrojado en el ejemplo anterior, por lo que se está indicando que es poco probable que los ojos se encuentren situados en esta nueva posición.

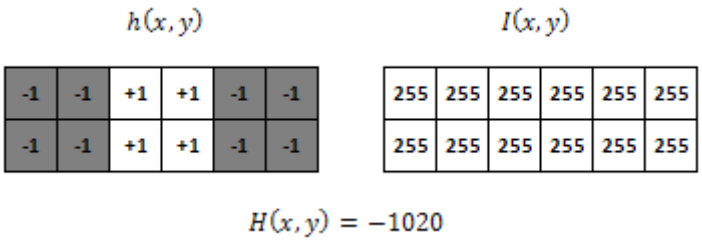


Figura 2.7: *Ejemplo de una característica Haar para detectar ojos aplicada sobre la región de la imagen de la frente.*

2.1.2.4. Imagen Integral

Como se ha podido comprobar, las características *Haar* son muy sencillas de calcular, pero aún así, suponen un coste computacional ligeramente alto. Hay que pensar que en una detección cada característica *Haar* utilizada tiene que evaluar toda la imagen pixel por pixel y, además, para diferentes escalas de la imagen. En definitiva esto es un una limitación para su uso en aplicaciones en tiempo real.

Con esta idea de mejorar la velocidad de procesamiento de las características *Haar*, Viola y Jones introdujeron por primera vez el concepto de **imagen integral [3]**. La imagen integral es una versión transformada de la imagen original que se crea previamente para su evaluación con las características *Haar*. Esta transformación consiste en crear una matriz, llamada imagen integral $II(x,y)$, que contiene en cada elemento (x,y) la suma de todos los pixeles de la imagen original $I(x,y)$ que queden por encima y a la izquierda de dicho punto, inclusive:

$$II(x,y) = \sum_{i \le x} \sum_{j \le y} I(i,j) \tag{2.2}$$

Usando la imagen integral, cualquier suma rectangular de pixeles puede ser calculada rápidamente con cuatro referencias. Esto se puede comprobar con un ejemplo. En la ***Figura 2.8*** se tiene una imagen de la cual se quiere calcular la suma de los pixeles de la región **D**. El valor de la imagen integral en la referencia **1** es la suma de los pixeles de la región **A**. El valor en la referencia **2** será **A+B**, en la referencia **3** será **A+C** y en la referencia **4** será **A+B+C+D**. De esta manera la suma de los píxeles de la región **D** se puede calcular como **4+1-(2+3)** sobre la imagen integral.

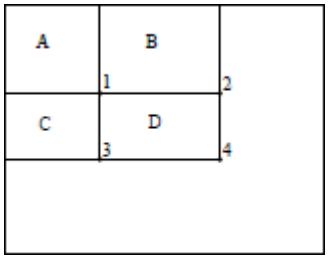


Figura 2.8: *Cálculo de la suma de pixeles de una región rectangular a partir de la imagen integral.*

Para el cálculo de la imagen integral se utilizan las dos siguientes recurrencias:

$$S(x,y) = S(x,y-1) + I(x,y) \tag{2.3}$$

$$II(x,y) = II(x-1,y) + S(x,y) \tag{2.4}$$

donde $S(x,y)$ es la suma de los pixeles de la fila x hasta la columna y, con $S(x,-1) = 0$ e $II(-1,y) = 0$. De esta manera la imagen integral se puede calcular en un solo paso sobre la imagen original.

En definitiva, la imagen integral es una técnica que permite evaluar las características *Haar* sobre una imagen de forma rápida, en cualquier posición y escala, utilizando pocas operaciones y en un tiempo constante.

2.1.3. Características *LBP*

2.1.3.1. Definición

En el trabajo presentado por He y Wang en **[5]** se describe por primera vez una nueva técnica para el análisis de texturas en imágenes a través de las llamadas *unidades de textura*. Esta técnica consiste en evaluar la disposición espacial y el contraste de los pixeles en pequeñas regiones de la imagen. Como resultado se obtiene un espectro de textura por cada región que permite identificar el tipo de textura global de la imagen. Desafortunadamente esta técnica no tuvo ninguna relevancia ni teórica ni práctica, por lo menos en un principio.

Unos años más tarde fue cuando Ojala et al. en **[6]** utilizaron la idea de las unidades de textura, pero dándoles una nueva visión. En este caso las unidades de textura son codificadas mediante un número binario de tal manera que cada tipo de textura pueda ser identificado fácilmente. A este nuevo método lo llamaron **características *LBP (Local Binary Patterns)***.

El operador *LBP* funciona de la siguiente manera. Sobre entornos de la imagen de 3x3 pixeles, se compara la intensidad de cada uno de los ocho pixeles vecinos con la intensidad del pixel central. Si la intensidad del pixel vecino es mayor o igual que la del pixel central, se le asigna a la posición del pixel vecino un valor de 1 o, en caso contrario, un valor de 0. Una vez comparados todos los pixeles, se tendrá un conjunto de ceros y unos que representarán un número binario. Multiplicando cada posición del número binario por su peso en decimal y sumando todo, se obtiene el valor de la característica *LBP* con el que se etiquete al pixel central. En la siguiente figura se muestra un ejemplo para comprender mejor este proceso:

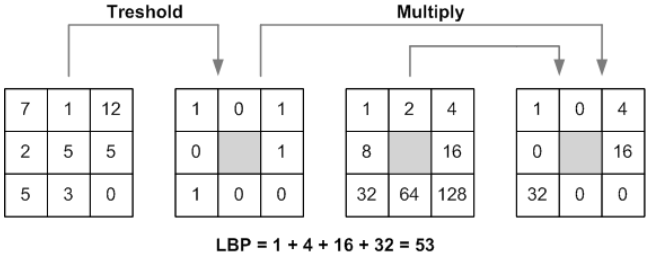


Figura 2.9: *Funcionamiento del operador LBP.*

Una descripción más formal del operador *LBP* puede darse mediante la siguiente expresión:

$$LBP(x_c, y_c) = \sum_{p=0}^{P-1} s(I_p - I_c)\, 2^p \tag{2.5}$$

con (x_c, y_c) la posición del pixel central , P el número de pixeles de la vecindad, I_p la intensidad de los pixeles vecinos, I_c la intensidad del pixel central y s la siguiente función:

$$s(x) = \begin{cases} 1, & x \geq 0 \\ 0, & si\ no \end{cases} \tag{2.6}$$

En definitiva, para cada pixel de la imagen se obtiene un valor asociado que indica el tipo de textura que presentan sus 8 pixeles vecinos. Con esta combinación de valores, a lo máximo se podrán representar 2^8 o 256 tipos de unidades de texturas. Así pues, calculando el histograma de la característica *LBP* y analizando sus valores, se puede extraer un patrón identificativo de texturas que permitan clasificar la imagen (ver ***Figura 2.10***).

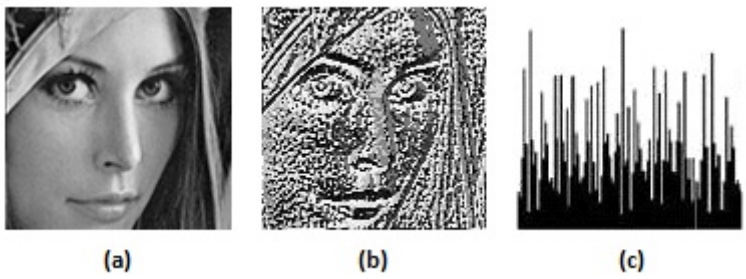


Figura 2.10: *Proceso de cálculo de la característica LBP de una imagen. (a) Imagen a evaluar. (b) Característica LBP. (c) Histograma de la característica LBP.*

Este mismo procedimiento puede ser aplicado para la detección de objetos. Si se conociese el patrón de texturas que presenta el objeto, sería tan fácil como calcular la característica *LBP* de la imagen donde se está buscando, obtener su histograma y encontrar su correspondencia por comparación.

2.1.3.2. *LBP* Circular

Después de unos años de su publicación original, Ojala et. al en **[7]** presentaron una nueva forma del operador *LBP* llamada ***LBP* circular**. A diferencia de la primera versión del operador *LBP* que era aplicable únicamente en una vecindad de 8 pixeles en entornos de 3x3 pixeles, esta nueva versión no pone ninguna limitación en cuanto al número de pixeles de la vecindad ni al tamaño del entorno, permitiendo identificar texturas en cualquier escala.

La idea que se propone ahora es considerar que los pixeles vecinos están sobre un entorno circular variable. Para poder describir este nueva forma, se tienen que definir dos nuevos parámetros que son el número de píxeles vecinos que van a componer el entorno y el radio o distancia a la que están situados estos vecinos. En lo siguiente la nomenclatura del operador se escribirá como $LBP_{P,R}$ siendo *P* el número de pixeles que componen la vecindad y *R* el radio del circulo de vecindad. En la siguiente imagen e muestran algunos ejemplos de vecindades circulares que se pueden aplicar:

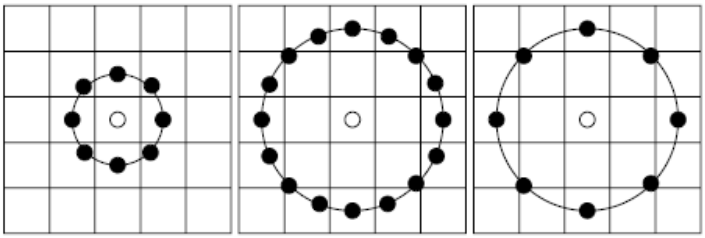


Figura 2.11: *Ejemplos de vecindades circulares. (a) Vecindad de 8 pixeles en un radio de 1 pixel. (b) Vecindad de 16 pixeles en un radio de 2 pixeles. (c) Vecindad de 8 pixeles en un radio de 2 pixeles.*

Las coordenadas de los pixeles del círculo de vecinos (x_p, y_p) se calculan a través de las siguientes expresiones:

$$x_p = x_c + Rcos\left(\frac{2\pi p}{P}\right) \tag{2.7}$$

$$y_p = y_c - Rsen\left(\frac{2\pi p}{P}\right) \tag{2.8}$$

con (x_c, y_c) las coordenadas del pixel central, R y P el radio y el número de pixeles del circulo de vecinos, y p el número de pixel vecino. Si las coordenadas resultantes no tienen ninguna correspondencia con coordenadas de la imagen, dichos pixeles son calculados por aproximación mediante una interpolación bilineal:

$$f(x,y) = \begin{bmatrix} 1-x & x \end{bmatrix} \begin{bmatrix} f(0,0) & f(0,1) \\ f(1,0) & f(1,0) \end{bmatrix} \begin{bmatrix} 1-y \\ y \end{bmatrix} \tag{2.9}$$

2.1.3.3. *LBP* Uniforme

Otra extensión del operador *LBP* original que se propone también en **[7]** son los llamados ***LBP* uniformes**, utilizados porque reducen el tamaño del set de patrones binarios y porque permiten implementar un descriptor invariante a la rotación. Esta extensión está inspirada en el hecho de que algunos patrones binarios aparecen con mayor frecuencia que otros en la textura de las imágenes.

Un *LBP* se llama uniforme si el patrón binario contiene a lo sumo dos transiciones de 0 a 1 o viceversa. Por ejemplo, los patrones binarios 00000000 (0 transiciones) y 01110000 (2 transiciones), son uniformes mientras que los patrones 11001001 (4 transiciones) y 01010101 (7 transiciones), no los son. Ahora la forma de representar que se está utilizando un *LBP* uniforme de 2 transiciones en un entorno circular será escribiendo $LBP_{P,R,U2}$.

El procedimiento de cálculo de los *LBP* uniformes será el mismo que el seguido hasta ahora, salvo que a los patrones no-uniformes se les asignará la misma etiqueta para todos. Por ejemplo, en una vecindad circular $LBP_{8,R}$ se pueden identificar un total de 256 patrones de los cuales 58 serán uniformes y el resto no. Así que, se asignarán 58 etiquetas diferentes para los patrones uniformes y al resto se le asignará la misma etiqueta que será la número 59.

Las razones de omitir los patrones no uniformes son fundamentalmente dos. La primera es que según los experimentos realizados por Ojala et al. sobre texturas en imágenes, casi el 90% de los patrones en una vecindad (8,1) son uniformes y alrededor del 70% en una vecindad (16,2). Esto demuestra que los patrones no-uniformes casi no se utilizan, así que se estarían malgastando etiquetas en el caso de usarlos. La otra razón es que los patrones uniformes estadísticamente dan mejores resultados en aplicaciones de reconocimiento. Una de las causas de ello es que ofrecen una mayor estabilidad ante la presencia de ruido. La otra causa es que se necesitan de menos muestras para el reconocimiento por lo que cualquier estimación siempre va a ser más fiable.

Otra de las ventajas aparte que ofrecen los *LBP* uniformes, es que permiten identificar características tales como puntos, líneas, bordes, directamente sobre el patrón binario. En la ***Figura 2.12*** se muestran algunos ejemplos. Los puntos negros representan un 0 en el patrón binario y los blancos un 1.

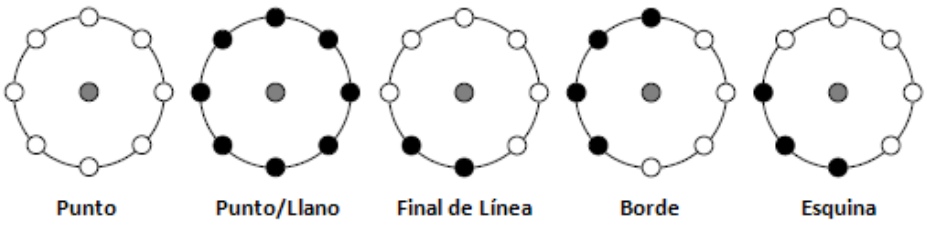


Figura 2.12: *Ejemplos de representación de características mediante patrones binarios.*

Además de todas estas variantes del operador *LBP*, en el artículo **[7]** se describen otras que incorporan más propiedades y funcionalidades, como por ejemplo la invariancia a la escala de grises o la invariancia a la rotación, pero que en definitiva se basan en la misma idea.

2. 2. Clasificadores

2.2.1. Introducción

Hasta este punto se ha visto como extraer las características de un objeto en un imagen a partir de dos técnicas: las características *Haar* y las características *LBP*. Pero ahora se plantea la cuestión de qué hacer con estas características. Como ya se explicó antes los objetos tienen una gran cantidad de características que los definen pero no todas serán de utilidad. Existirán aquellas que sean comunes a varios objetos o aquellas que según en qué condiciones cambien. En definitiva, lo que interesa en las aplicaciones de detección es identificar solamente aquellas que distingan al objeto del resto del entorno de la imagen incluso bajo distintas condiciones.

Pues bien, el siguiente paso será explicar cómo a partir de las características obtenidas con las técnicas citadas anteriormente, se seleccionan aquellas que permiten identificar al objeto a detectar. Esta tarea la realizarán los llamados **clasificadores**. Para ser más exacto, se describirá un tipo de clasificadores que son los denominados **clasificadores en cascada**, ya que son el tipo utilizado por las librerías de *OpenCV* para trabajar con características *Haar* y *LBP*.

2.2.2. Clasificadores *Weak*

La primera idea que se puede ocurrir para definir un clasificador sería algo tan sencillo como elegir un valor umbral, de tal manera, que si el valor de la característica calculada está por encima de ese umbral se considerará que pertenece al objeto a detectar, o en caso contrario, no lo hará. A este sistema de catalogar las características se le llama **clasificador *weak (weak classifier)***.

El clasificador *weak* más sencillo que se puede construir es el denominado **clasificador *stump (stump classifier)***. Está definido por una función $h(x)$ que depende de: el valor de una característica f aplicada sobre una región x, un umbral t y una paridad p que representa el sentido de la desigualdad. La salida del clasificador es un valor binario que indica si el valor de una característica está por encima o por debajo del valor umbral.

$$h(x) = \begin{cases} 1, & pf(x) > pt \\ 0, & si\ no \end{cases} \tag{2.10}$$

La dificultad en este tipo de clasificadores es encontrar el valor umbral t que permita clasificar correctamente el mayor número de características posible. Para ello, existe una gran variedad de algoritmos que encuentran el umbral óptimo a partir del entrenamiento de una serie de muestras del objeto a detectar.

Este tipo de clasificadores se pueden ver como un nodo de decisión en el que dependiendo del valor de la característica $f(x)$, el proceso de clasificación sigue un camino u otro:

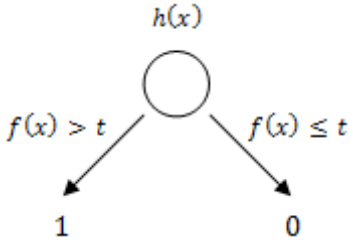


Figura 2.13: *Esquema de un clasificador stump.*

El principal problema que presentan estos clasificadores es que por sí solos no son capaces detectar objetos con una tasa de aciertos aceptable, por lo que en aplicaciones reales no se suelen utilizar solos.

2.2.3. Clasificadores *Strong*

Este tipo de clasificadores vienen a solucionar el problema de los anteriores. La idea consiste en agrupar varios clasificadores *weak*, de tal forma que en conjunto se consigue reducir considerablemente el error en la clasificación. A este tipo de clasificador se le llama **clasificador *strong (strong classifier)***.

Una de las formas de crear un clasificador *strong* es mediante la combinación lineal de varios clasificadores *weak*. A esta forma de clasificador se le denomina concretamente ***perceptrón (perceptron)***. En base a esta idea, el valor de un clasificador *strong* $H(x)$ se puede expresar como:

$$H(x) = \begin{cases} 1, & pF(x) > pT \\ 0, & si\ no \end{cases} \tag{2.11}$$

$$F(x) = \sum_{i=1}^{N} w_i h_i(x) \tag{2.12}$$

donde $h_i(x)$ representa a los clasificadores *weak*, w_i el peso de cada clasificador, N el número de clasificadores, T el umbral y p el sentido de la desigualdad. En este caso, además del umbral de cada clasificador *weak* t_i, también se tendrán que ajustar sus pesos w_i y el umbral del clasificador *strong* T, mediante algún tipo de algoritmo de entrenamiento de los que existen.

Además de este tipo de clasificador *strong*, existen otros con los que se consigue el mismo objetivo: clasificadores bayesianos, clasificadores borrosos, arboles de decisión, máquinas de vectores de soporte, redes neuronales, etc. Es cuestión del diseñador elegir el que mejor se amolde a sus especificaciones.

En la siguiente imagen se muestra un esquema que representa el funcionamiento del clasificador perceptrón:

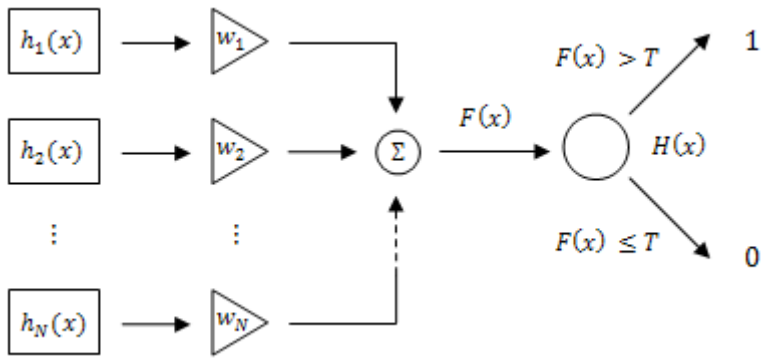


Figura 2.14: *Esquema de un clasificador perceptrón.*

Aunque el clasificador *strong* mejora los resultados frente a los clasificadores *weak*, puede que según para que aplicaciones aún no sea suficiente y se necesite de clasificadores aún más complejos.

2.2.4. Clasificadores en Cascada

2.2.4.1. Planteamiento Inicial

El principio para detectar objetos, descrito por Viola y Jones en **[3]**, surge de analizar las propiedades de las características *Haar*. Como ya se explicó, cada tipo de característica *Haar* está diseñado para extraer un determinado rasgo del objeto. Pero en el proceso de detección, con un solo rasgo no sería suficiente para distinguir el objeto. Y, por qué no combinar el efecto de varias características *Haar*. Así, se recogerían varios rasgos del objeto y la detección sería mucho más sencilla.

En base a esto, lo que plantean Viola y Jones es formar un clasificador *strong* a partir de la combinación de varios clasificadores *weak*, en cada uno de los cuales se evalúa un tipo de característica *Haar*. Las características que participan en el clasificador serán seleccionadas y ponderadas por el algoritmo de entrenamiento en base a las puntuaciones que hayan arrojado tras evaluar un set de muestras positivas (en las que aparece el objeto) y negativas (en las que no aparece el objeto).

Para este clasificador, el funcionamiento del detector consistiría en ir evaluando ventanas de la imagen de un determinado tamaño, para distintas escalas de la propia imagen y para cada una de las características *Haar* del clasificador. Si el clasificador determina que las características encontradas en la ventana son las del objeto a detectar, dicha ventana se clasificará como positiva, si no, se clasificará como negativa; y así hasta evaluar todo el conjunto de ventanas de la imagen.

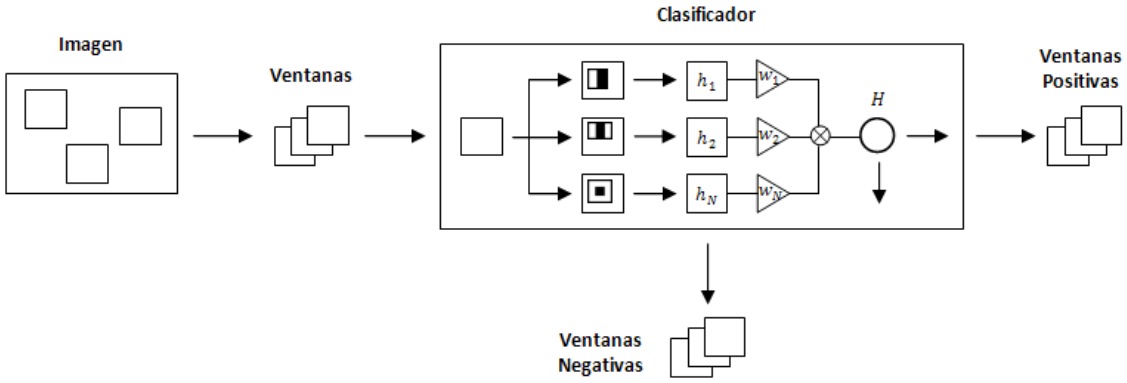


Figura 2.15: *Esquema de funcionamiento del detector de objetos con un único clasificador strong.*

Tras los ensayos realizados, Viola y Jones se dieron cuenta de que esta manera de detectar objetos era poco eficiente. Comprobaron que el tiempo de cómputo del detector, en su mayoría, se invertía en evaluar ventanas de la imagen donde no se encuentra el objeto. Esto es debido a que comúnmente el espacio de la imagen que representa el fondo es bastante mayor que él del objeto. Si a esto se suma la cantidad de ventanas que puede contener el fondo, que las ventanas se evalúan para distintas escalas de la imagen, y que para cada una de ellas se aplican los distintos tipos de características *Haar* del clasificador; definitivamente el tiempo que se está perdiendo en clasificar el fondo puede ser significante.

2.2.4.2. Descripción

En busca de mejorar el rendimiento en la detección, reduciendo el tiempo de cómputo, Viola y Jones proponen una alternativa en el mismo artículo **[3]**. Consiste en combinar clasificadores *strong* en una estructura en cascada o árbol jerárquico. A este nuevo tipo de clasificador le denomina **clasificador en cascada**.

La idea surge del hecho de que dentro de una imagen, es más fácil y rápido determinar dónde no se encuentra el objeto buscado, que dónde sí lo hace. Con este razonamiento, el clasificador en cascada está formado por clasificadores *strong* dispuestos uno detrás de otro formando etapas, de tal manera que cada etapa es más compleja que la anterior. El propósito es que las primeras etapas se destinen a rechazar las ventanas de la imagen correspondientes al fondo y las últimas etapas se encarguen de evaluar las ventanas donde posiblemente esté el objeto.

Por tanto, con este clasificador se consigue ahorrar el tiempo que se requeriría en evaluar las ventanas del fondo en un único clasificador complejo. Ahora, esta tarea la realizan las primeras etapas del clasificador en cascada, formadas por clasificadores muy simples y por tanto rápidos de aplicar.

En cuanto al funcionamiento del clasificador, será el mismo que el explicado en el planteamiento inicial. La diferencia reside en que en cada etapa se deciden las ventanas que pasan a la siguiente y cuáles se rechazan directamente. Para que una ventana pueda considerarse como positiva deberá pasar todas las etapas del clasificador en cascada.

En la siguiente figura se muestra el esquema de funcionamiento del clasificador en cascada:

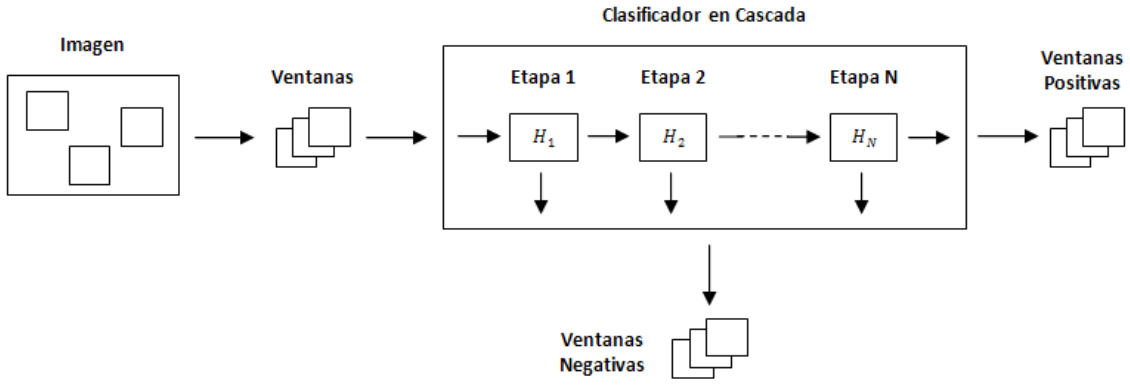


Figura 2.16: *Esquema de funcionamiento del detector de objetos con un clasificador en cascada.*

Finalmente Viola y Jones llegaron a la conclusión de que éste era el mejor diseño de clasificador entre los dos propuestos en **[3]**. En los test realizados, la estructura en cascada resultó ser en promedio mucho más rápido que el basado en un único clasificador *strong*, manteniendo aún así la tasa de aciertos.

2.2.4.3. Mejoras

Tras el éxito del clasificador en cascada, Lienhart et al. en **[8]** deciden llegar aún más lejos añadiendo un grado más de complejidad al clasificador. Lo que proponen es sustituir la combinación de clasificadores *weak* que componían hasta ahora cada etapa (o clasificador *strong*), por una combinación de arboles de clasificadores *weak* (ver ***Figura 2.17***). A esta forma de agrupar clasificadores *weak* se le denomina **arboles de decisión *CART (Classification and Regression Tree)***.

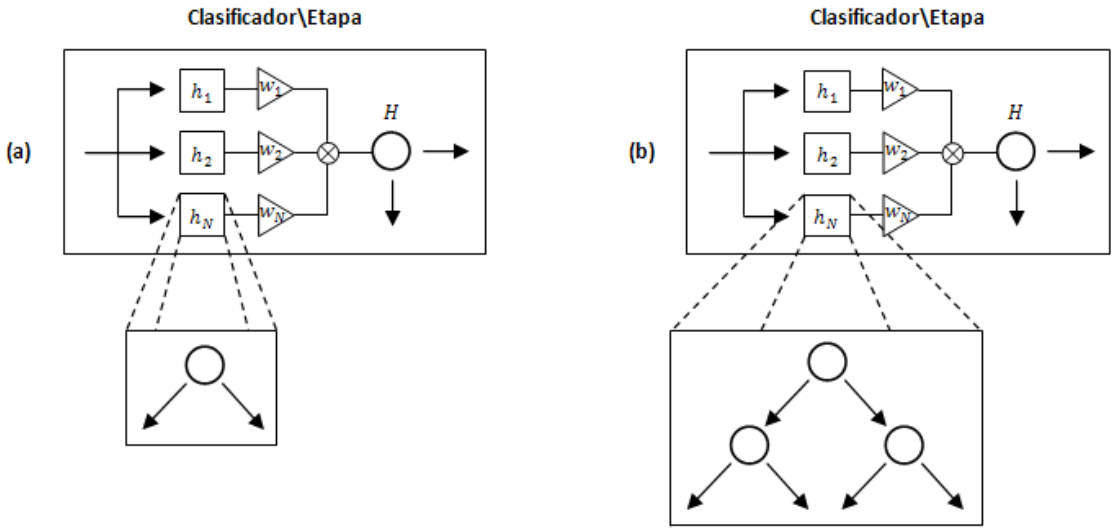


Figura 2.17: *Estructura de una etapa del clasificador en cascada. (a) Con clasificadores weak. (b) Con clasificadores CART.*

Un *CART* se define como un modelo de predicción formado por construcciones lógicas o reglas encadenadas que sirven para clasificar una serie de condiciones que ocurren de forma sucesiva. La estructura de un *CART* la forman varios nodos de decisión enlazados entre sí formando un árbol, en cada uno de los cuales se evalúa una condición. El proceso de clasificación comienza con el primer nodo (nodo raíz). Dependiendo de si la condición se cumple o no, se pasa a los siguientes nodos (nodos hijos) por un camino u otro (ramas). En el siguiente nodo se repite el proceso y así hasta llegar a un nodo terminal (nodo hoja) que definirá el tipo de clasificación.

Extrapolando esta descripción, en un clasificador *CART* los nodos de decisión vienen representados por clasificadores *weak* $h_i(x)$, y las condiciones en cada nodo vienen determinadas por el valor de las características *Haar* $f_i(x)$ y los umbrales t_i (ver ***Figura 2.18***). Se entiende que en cada nodo se evalúa un tipo de característica *Haar* distinto.

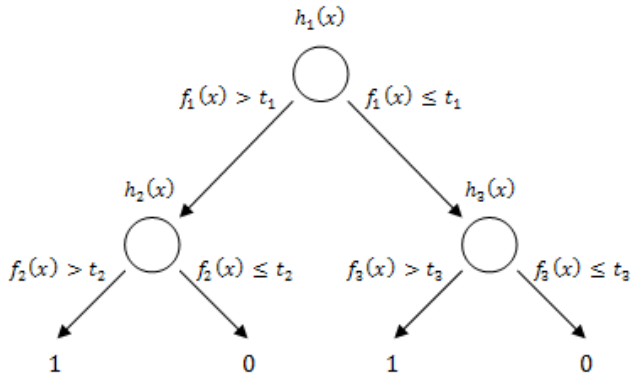


Figura 2.18: *Estructura de un clasificador CART.*

La estructura de estos árboles no viene predefinida y luego se ajusta en el proceso de entrenamiento, sino que es el propio algoritmo de entrenamiento el encargado de construirla. Comienza seleccionando el nodo raíz y poco a poco va añadiendo ramas y nodos (*splitting*) hasta cumplir con unos objetivos de detección. Generalmente, para que no se creen arboles de decisión muy profundos se fija el número de nodos máximo que el algoritmo puede utilizar.

El motivo por el que Lienhart y sus compañeros introdujeron los *CART* en el clasificador en cascada, es porque comprobaron que, en la versión de Viola y Jones, a medida que aumentaba la complejidad del objeto a detectar, también lo hacia el número de características necesarias para clasificarlo y, por tanto, el tiempo de computo. Los clasificadores *CART* tienen la ventaja de poder hacer clasificaciones muy difíciles con pocas características, debido a esas dependencias que se crean entre ellas, con lo cual el rendimiento del clasificador se ve notablemente mejorado.

2.3. Entrenamiento de Clasificadores

2.3.1. Introducción

Para que un clasificador sea capaz de detectar un objeto en concreto, es necesario que se le someta a un proceso de entrenamiento en el que aprenda a distinguir las características importantes que van a definir a ese objeto. Esto no puede hacerse de otra manera que no sea haciéndole procesar un conjunto amplio de muestras de dicho objeto. Este proceso conlleva tanto la formación del propio clasificador como el ajuste de sus parámetros hasta conseguir unos objetivos de detección fijados. Ahora bien, debe haber algún mecanismo que se encargue de realizar todas estas tareas. Ese mecanismo son los llamados **algoritmos de entrenamiento** (o también conocidos como algoritmos de aprendizaje).

Las librerías de *OpenCV* implementan varios tipos de algoritmos de aprendizaje para los clasificadores en cascada. El objetivo de este apartado será pues explicar los conceptos básicos y características de estos algoritmos con el fin de entender, cómo funcionan y así poder trabajar con ellos. El primer concepto que habrá que entender es el de ***Boosting*** que no es más que la metodología base de entrenamiento seguida por estos algoritmos. Además se verá la primera aplicación práctica de *Boosting*, que es el algoritmo llamado ***AdaBoost***. Después, se continuará viendo los **tipos de *Boosting*** que se han ido desarrollando con el tiempo y las características principales de cada uno de ellos. Y para terminar se describirá el caso especial de *Boosting* aplicado al **entrenamiento de un clasificador en cascada**.

2.3.2. *Boosting*

El ***Boosting*** es uno de los desarrollos más importantes en el ámbito de la clasificación. Fue definido por primera vez por Schapire en sus investigaciones **[9]**. La idea que proponía era crear una nueva metodología que permitiese entrenar clasificadores *strong* formados por estructuras de clasificadores *weak*.

Al igual que un clasificador *strong* es una formación de clasificadores *weak*, el *Boosting* se compone de pequeños algoritmos de aprendizaje llamados ***weak learning***. Estos pequeños algoritmos son los encargados del entrenamiento a bajo nivel, es decir, de cada uno de los clasificadores *weak* por separado. Supervisando estas tareas desde un nivel más alto, se encuentra lo que se podría llamar el ***strong learning*** que gestiona los resultados de cada *weak learning* y va formando lo que sería en definitiva el clasificador *strong*.

La característica principal que distingue al *Boosting* de otro tipo de metodologías de aprendizaje, es que la estructura de clasificadores *weak* se va construyendo de forma iterativa, de tal forma que cada clasificador corrige el error de clasificación de su predecesor sobre las muestras de entrenamiento. De esta manara, se consigue que el conjunto de clasificadores *weak* tenga una buena precisión.

En el cuadro de la ***Figura 2.19*** se define el esquema completo de aplicación del primer método *Boosting* ideado por Schapire:

Entrada:

- $(x_1, y_1), \dots, (x_N, y_N)$, con x_i las muestras de entrenamiento, y_i la clase de las muestras, y N el número de muestras.

Pasos:

1. Entrenar un clasificador *weak* $h_1(x)$ con un conjunto de muestras $X_1 < N$ seleccionadas de forma aleatoria.
2. Entrenar un clasificador weak $h_2(x)$ sobre un conjunto de muestras $X_2 < N$, con la mitad de las muestras mal clasificadas por el clasificador weak $h_1(x)$.
3. Entrenar un último clasificador $h_3(x)$ con las muestras mal clasificadas en $h_1(x)$ y $h_2(x)$.
4. Obtener el clasificador *strong* como $H(x) = \sum_{m=1}^{3} h_m(x)$.

Salida:

- $H(x)$, clasificador *strong* cuyo signo $sign[H(x)]$ indica la clase clasificada.

Figura 2.19: *Cuadro resumen del algoritmo Boosting.*

Decir que el *Boosting* no es una metodología cerrada, sino que admite muchas variantes en la forma de aplicarlo. La forma más utilizada y significante de *Boosting* es el llamado ***AdaBoost (Adaptative Boosting)*** creado por Freund y Schapire en **[10]**. Este algoritmo fue el primero en poder integrar los clasificadores *weak* en el proceso de *Boosting*. Se dice que es adaptativo en el sentido de que la secuencia de construcción de clasificadores *weak* se va ajustando en favor de las muestras mal clasificadas por los clasificadores *weak* creados previamente.

El proceso de entrenamiento mediante el algoritmo *AdaBoost* es un proceso que se realiza de forma iterativa. El método comienza con un set de imágenes de muestra positivas (en las que aparece el objeto a detectar) y negativas (en las que no aparece), a las que, en un principio, se les asigna el mismo peso. En cada iteración, el set de muestras es utilizado por el *weak learning* para entrenar al conjunto de clasificadores *weak* asociados a cada característica. De los entrenados, se elije el clasificador *weak* con menor error, se le asigna un peso cuyo valor indica la precisión que ha arrojado y se añade al clasificador *strong*. En este punto, se actualizan los pesos de las muestras, aumentando el de las mal clasificadas y disminuyendo el de las bien clasificadas. De este modo, se fuerza a que al algoritmo se centre en las muestras más difíciles de clasificar, con lo que el resultado final será más robusto. Llegado aquí, este proceso continúa una y otra iteración hasta que se cumplan unos objetivos de detección que generalmente suelen ser el alcanzar una tasa de acierto determinada.

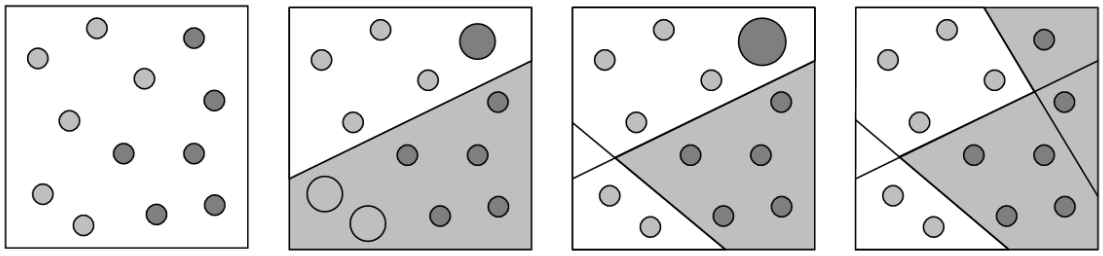


Figura 2.20: *Construcción de un clasificador strong a partir de clasificadores weak mediante Adaboost.*

La versión moderna y optimizada de este algoritmo es el denominado *Discrete AdaBoost*, que ha servido como base para el desarrollo de variantes, tales como el *Real AdaBoost*, el *LogitBoost* y el *Gentle AdaBoost*. La diferencia entre estos algoritmos suele encontrase en la forma de ajustar los clasificadores *weak* o los pesos de las muestras, por lo demás, se podría decir que son iguales ya que siguen el mismo esquema de funcionamiento. En el siguiente apartado se describirán con detalle el esquema de procedimiento por estos algoritmos.

2.3.3. Tipos de *Boosting*

2.3.3.1. *Dicrete AdaBoost*

Como ya se ha comentada, el ***Discrete AdaBoost*** es la versión mejorada del algoritmo original *AdaBoost* y fue formulada por Freud y Schapire en **[11]**. El adjetivo "discreto" proviene del hecho de que está diseñado para que sus clasificadores *weak* solo puedan dar dos valores (-1 o +1) que indican las dos posibles clases a clasificar, frente a otros tipos de *Boosting* que permiten dar un rango más amplio de valores aunque sigan clasificando solo dos clases.

En el siguiente cuadro, se muestra el proceso de entrenamiento con *Discrete Adaboost*:

Entrada:

- $(x_1, y_1), \dots, (x_N, y_N)$, con x_i las muestras de entrenamiento, y_i la clase de muestras con $y_i = \{-1, +1\}$, y N el número de muestras.
- M, máximo número de iteraciones o de clasificadores *weak*.

Pasos:

1. Inicializar los pesos de los datos $w_i = \frac{1}{N}$ con $i \in \{1, \dots, N\}$.
2. Para $m = 1, \dots, M$:
   a. Ajustar el clasificador *weak* $h_m(x) \in [-1, +1]$ usando los pesos w_i sobre los muestras de
   b. Obtener el error de clasificación $\epsilon_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq h_m(x_i))}{\sum_{i=1}^{N} w_i}$
   c. Calcular los pesos de los clasificadores *weak* $\alpha_m = \frac{1}{2} \log\left(\frac{1-\epsilon_m}{\epsilon_m}\right)$.
   d. Actualizar los pesos $w_i \leftarrow w_i \exp\left(-\alpha_m I(y_i \neq h_m(x_i))\right)$ y normalizarlos para que $\sum_i w_i = 1$.
3. Obtener el clasificador *strong* como $H(x) = \sum_{m=1}^{M} \alpha_m h_m(x)$.

Salida:

- $H(x)$, clasificador *strong* cuyo signo $sign[H(x)]$ indica la clase clasificada.

Nota:

- $\mathrm{I(c)}$ es una función indicadora de la clase clasificada, con $\mathrm{I(c)} = 1$ si $\mathrm{c} =$ "true" y $\mathrm{I(c)} = 0$ si $\mathrm{c} =$ "false".

Figura 2.21: *Cuadro resumen del algoritmo Discrete AdaBoost.*

En resumen, el clasificador *strong* final es una suma ponderada de clasificadores *weak* entrenados mediante un conjunto de muestras ponderadas en cada iteración. Tanto los pesos de las muestras como los pesos de los clasificadores *weak* son función del error de clasificación que cada uno ha arrojado durante el proceso de entrenamiento.

2.3.3.2. *Real AdaBoost*

El algoritmo ***Real AdaBoost*** (Schapire y Singer en **[12]**) es una modificación del algoritmo *Discrete AdaBoost* que busca mejorar el proceso de aprendizaje. En este caso se pretende que los valores de salida de los clasificadores *weak* tenga un significado más allá que el de solo clasificar. Con esa idea, en este algoritmo se calculan los clasificadores *weak* para que el valor de su salida se corresponda en signo con la clase clasificada y en módulo con la probabilidad de que la muestra pertenezca a esa clase. Como ahora el valor de salida es un conjunto de valores reales y no discretos, se le da el adjetivo de "real".

El proceso de aplicación del algoritmo se explica a continuación:

Entrada:

- $(x_1, y_1), \dots, (x_N, y_N)$, con x_i las muestras de entrenamiento, y_i la clase de muestras con $y_i = \{-1, +1\}$, y N el número de muestras.
- M, máximo número de iteraciones o de clasificadores *weak*.

Pasos:

1. Inicializar los pesos de las muestras: $w_i = \frac{1}{N}$ con $i \in \{1, \dots, N\}$.
2. Para $m = 1, \dots, M$:
   a. Ajustar la probabilidad de la clase estimada $p_m(x) = P_m(y = 1|x) \in [0,1]$, usando los pesos w_i sobre los muestras de entrenamiento.
   b. Obtener el clasificador *weak* como $h_m(x) = \frac{1}{2} log\left(\frac{1-p_m(x)}{p_m(x)}\right) \in R$.
   c. Actualizar los pesos $w_i \leftarrow w_i \exp\left(-y_i H_m(x_i)\right)$ y normalizarlos para que $\sum_i w_i = 1$.
3. Obtener el clasificador *strong* final como $H(x) = \sum_{m=1}^{M} h_m(x)$.

Salida:

- $H(x)$, clasificador *strong* cuyo signo $sign[H(x)]$ indica la clase clasificada.

Figura 2.22: *Cuadro resumen del algoritmo Real AdaBoost.*

Comparando el *Real AdaBoost* con el *Discrete Adaboost*, se ve que la diferencia radica en que los clasificadores *weak* del primero se ajustan respecto al error de clasificación mientras que los del segundo lo hacen respecto a la probabilidad de clasificación.

2.3.3.3. *LogitBoost*

El ***LogitBoost (Logistic Boosting)***, formulado por Friedman et al. en **[13]**, es otro tipo de *Boosting* que implementa una sistema de entrenamiento distinto al visto en los diferentes algoritmos *AdaBoost*. La idea propuesta consiste en interpretar el método *Boosting* como si de un problema de ajuste de funciones se tratara. En concreto, los clasificadores *weak* son vistos como funciones de regresión "logística" y son calculados haciendo uso del algoritmo de ajuste de funciones de Newton sobre las muestras de entrenamiento ponderadas, esta vez, por mínimos cuadrados.

El siguiente cuadro describe con detalle los pasos seguidos por este algoritmo:

Entrada:

- $(x_1, y_1), \dots, (x_N, y_N)$, con x_i las muestras de entrenamiento, y_i la clase de muestras con $y_i = \{-1, +1\}$, y N el número de muestras.
- M, máximo número de iteraciones o de clasificadores *weak*.

Pasos:

1. Inicializar los pesos $w_i = \frac{1}{N}$ con $i = 1, \dots, N$, el valor del clasificador *strong* $H(x) = 0$, y la probabilidad estimada $p(x_i) = \frac{1}{2}$.
2. Para $m = 1, \dots, M$:
   a. Calcular la respuesta de trabajo $z_i = \frac{y_i - p(x_i)}{p(x_i)(1-p(x_i))}$ y los pesos $w_i = p(x_i)\big(1 - p(x_i)\big)$.
   b. Ajustar el clasificador *weak* $h_m(x) = H(x) + h_m(x)$ ponderando las muestras con los pesos w_i por mínimos cuadrados.
   c. Actualizar el clasificador $H(x) \leftarrow H(x) + \frac{1}{2} h_m(x)$ y la probabilidad $p(x) \leftarrow \frac{e^{H(x)}}{e^{H(x)} + e^{-H(x)}}$.
3. Obtener el clasificador *strong* $H(x) = \sum_{m=1}^{M} h_m(x)$.

Salida:

- $H(x)$, clasificador *strong* cuyo signo $sign[H(x)]$ indica la clase clasificada.

Figura 2.23: *Cuadro resumen del algoritmo LogitBoost.*

2.3.3.4. *Gentle AdaBoost*

Otra versión de algoritmo *AdaBoost* es el llamado ***Gentle AdaBoost***. Este algoritmo fue desarrollado también por Friedman et al. en **[13]** y adapta algunas de las herramientas utilizadas en el algoritmo *LogitBoost*. Más específicamente, se trata de una versión del algoritmo *Real AdaBoost* que utiliza el método de Newton junto con la ponderación de muestras por mínimos cuadrados para ajustar los clasificadores *weak*. El término "gentle" es debido a que el algoritmo no difiere en mucho del *Real AdaBoost*.

Los pasos del algoritmo se describen en el siguiente cuadro:

Entrada:

- $(x_1, y_1), \dots, (x_N, y_N)$, con x_i las muestras de entrenamiento, y_i la clase de muestras con $y_i = \{-1, +1\}$, y N el número de muestras.
- M, máximo número de iteraciones o de clasificadores *weak*.

Pasos:

1. Inicializar los pesos de las muestras: $w_i = \frac{1}{N}$ con $i \in \{1, \dots, N\}$.
2. Para $m = 1, \dots, M$:
   a. Ajustar el clasificador *weak* $h_m(x)$ ponderando las muestras con los pesos w_i por mínimos cuadrados.
   b. Calcular el clasificador como $H(x) = H(x) + h_m(x)$.
   c. Actualizar los pesos $w_i \leftarrow w_i \exp\left(-y_i H_m(x_i)\right)$ y normalizarlos para que $\sum_i w_i = 1$.
3. Obtener el clasificador *strong* final como $H(x) = \sum_{m=1}^{M} h_m(x)$.

Salida:

- $H(x)$, clasificador *strong* cuyo signo $sign[H(x)]$ indica la clase clasificada.

Figura 2.24: *Cuadro resumen del algoritmo Gentle AdaBoost.*

En definitiva, se ha podido comprobar que todos los tipos de algoritmos vistos, aunque difieren comúnmente en la forma de ajustar los clasificadores y los pesos, el esquema de aplicación de cada uno de ellos es el mismo. Es por ello que las ventajas de utilizar uno u otro suelen ser mínimas y normalmente van a depender, más que del algoritmo utilizado, de otros parámetros como las características del objeto a detectar, el número y la calidad de las muestras, la tasa de aciertos y fallos permisibles, etc. Por esta razón se aconseja probar todos los tipos de algoritmos posibles y elegir el que mejor resultados ofrezca.

2.3.4. Entrenamiento de Clasificadores en Cascada

Además de describir la estructura de un clasificador en cascada, Viola y Jones en **[3]** también proponen un algoritmo de aprendizaje para entrenarla. El algoritmo que plantean surge de interpretar cada etapa de la cascada como un único clasificador *strong* independiente del resto. Desde este punto de vista, cada etapa puede ser entrenada de forma separada por cualquiera de los algoritmos de *Boosting* vistos. Ahora lo necesario es definir un algoritmo que trabaje por encima coordinando el entrenamiento de cada etapa.

Viola y Jones establecen que el algoritmo de entrenamiento de un clasificador en cascada va a estar determinado por dos parámetros: la **tasa de verdaderos positivos *(true positive rate)*** y la **tasa de falsos positivos *(false positive rate)***. La tasa de verdaderos positivos se define como la razón de muestras que siendo positivas se han clasificado como tal, mientras que la tasa de falsos positivos es la razón de muestras negativas que han sido clasificadas como positivas. En definitiva, estos dos parámetros son importantes porque van a definir los límites del

entrenamiento en función de la precisión que se pretenda alcanzar con el clasificador. Así pues, teniendo en cuenta que el clasificador en cascada está formada por etapas, las tasas de verdaderos positivos TPR y de falsos positivos FPR totales vendrán dadas por las siguientes expresiones:

$$TPR = \prod_{i=1}^{N} tpr_i \tag{2.13}$$

$$FPR = \prod_{i=1}^{N} fpr_i \tag{2.14}$$

Con N el número de etapas del clasificador, tpr_i y fpr_i, las tasas de verdaderos positivos y falsos positivos de cada una de las etapas por separado. Esta forma de descomponer las tasas va a permitir tratar las etapas de la cascada como clasificadores independientes. Esto facilitará las tareas del algoritmo de entrenamiento que ahora podrá entrenar cada una de las etapas por separado hasta conseguir las tasas parciales fijadas. Se sabe que manteniendo las tasas parciales, las tasas totales están aseguradas. En la siguiente figura se muestran los pasos seguidos por el algoritmo de entrenamiento para construir un clasificador en cascada:

Entrada:

- tpr_{min} y fpr_{max}, valores de la mínima tasa de verdaderos positivos y de la máxima tasa de falsos positivos aceptables por etapa.
- FPR_{max}, tasa de falsas alarmas del conjunto.
- P y N, conjunto de muestras de entrenamiento positivas y negativas.

Pasos:

- Inicializar tasas $TPR_0 = 1.0;\ FPR_0 = 1.0$ y el índice $i = 0$.
- Mientras $FPR_i > FPR_{max}$.
  1. Incrementar el índice $i = i + 1$, actualizar la tasa de falsos positivos $FPR_i = FPR_{i-1}$ e inicializar el número de clasificadores *weak* $n_i = 0$.
  2. Mientras $FPR_i > fpr_{max} \cdot FPR_{i-1}$.
     - Incrementar el número de clasificadores *weak* $n_i = n_i + 1$.
     - Usar P y N para entrenar un clasificador con n_i características mediante *Adaboost*.
     - Evaluar el clasificador en cascada actual sobre el conjunto de muestras de entrenamiento para determinar FPR_i y TPR_i.
     - Decrementar el umbral del clasificador i-ésimo hasta que el clasificador en cascada actual tenga una tasa de aciertos de al menos $tpr_{min} \cdot TPR_i$ (esto también afecta a FPR_i).
  3. $N \leftarrow \emptyset$
  4. Si $FPR_i > FPR_{max}$ entonces evaluar la cascada de clasificadores sobre el conjunto de muestras y colocar las falsas alarmas dentro del conjunto N.

Salida:

- C, cascada de clasificadores entrenados.

Figura 2.25: *Cuadro resumen del algoritmo de entrenamiento de un clasificador en cascada.*

Explicando de forma resumida lo mostrado en el esquemático, se puede comprobar que el algoritmo de entrenamiento de un clasificador en cascada es un proceso iterativo. En cada iteración, se construye una etapa de la cascada y se entrena, a través de cualquiera de los algoritmos *Boosting* vistos, sobre el conjunto de muestras introducido al algoritmo. Sobre el mismo conjunto de muestras se calculan las tasas de la etapa y se comparan con las tasas fijadas. Si se superan, se vuelve a ajustar esa etapa; si no se superan, se pasa a la siguiente etapa; y así hasta alcanzar las tasas totales establecidas.

Decir que éste es el primer algoritmo que se propuso para entrenar a un clasificador en cascada pero que con el tiempo se ha ido perfeccionando. Los algoritmos modernos, aunque han optimizado algunas partes, siguen manteniendo la misma estructura que el algoritmo original, por lo que entender cómo funcionan habiendo comprendido ésta no debería ser un problema.

2.4. Evaluación de Clasificadores

2.4.1. Introducción

Hasta este punto, se ha explicado lo qué es un clasificador, la forma de construir uno y como entrenarlo para que pueda detectar un tipo de objeto en concreto. Con estos pasos sería más que suficiente para poder empezar a trabajar con el clasificador, pero en aplicaciones reales es necesario dar un paso más. En este sentido, el proceso de diseño no estaría completo si no se realiza una **evaluación del clasificador** que permita dar a conocer el rendimiento que va a tener el clasificador.

Generalmente, el proceso de evaluación consiste en aplicar al clasificador un conjunto de muestras, obtener los resultados de clasificación que realiza y compararlos con los de la realidad. El análisis de los resultados de este proceso permitirá extraer información acerca del comportamiento del clasificador. Datos relacionados con el rendimiento, como la tasa de aciertos o la tasa de fallos, pueden ser algunas de las características del clasificador que se pueden calcular y que ayudarían a decidir si el clasificador cumple o no con los criterios de la aplicación donde vaya a implantarse. Pero también son interesantes otro tipo de características como la influencia de los parámetros de entrenamiento. Es fácil de comprender que parámetros como el tipo de características, el tipo de clasificador, el número de iteraciones, el número de muestras o las máximas tasas admisibles van a afectar directamente al comportamiento del clasificador. Evaluar los resultados del entrenamiento permitiría ajustar estos parámetros para conseguir el mejor clasificador posible.

Así pues, en este apartado se presentarán un conjunto de herramientas que permiten evaluar el comportamiento de los clasificadores vistos hasta ahora. Específicamente se verán la **matriz de confusión**, que es una forma visual de representar el número de aciertos y fallos del clasificador, y tres curvas gráficas que se derivan de la *matriz de confusión* y que permiten valorar el rendimiento de los clasificadores: las **curvas *ROC***, las **curvas *DET*** y las **curvas *PR***.

2.4.2. Matriz de Confusión

En el campo de la visión artificial, una de las herramientas más importantes que se utilizan para evaluar de forma visual los resultados del entrenamiento de un clasificador es la llamada **matriz de confusión [14]**. Este método surge como consecuencia de comparar la clasificación realizada por un clasificador sobre un conjunto de muestras, con la clase real a la que pertenecen esas muestras. Considerando un clasificador binario (dos clases), los dos posibles resultados que se pueden obtener son positivo (*P*) o negativo (*N*). Pero estas dos clases no tienen porque coincidir con la realidad, el clasificador no va a ser perfecto y presentará un cierto grado error, es decir, puede equivocarse. Esta es la razón por la que al trasladar la predicción del clasificador al ámbito real se tengan que crear nuevas clases que surjan de combinar las dos posibles clases dadas por el clasificador con el acierto o fallo en la clasificación.

En el entorno real, el resultado de clasificar un conjunto de muestras mediante un clasificador binario puede ser:

- **Verdadero Positivo *(True Positive) (TP)*.** Significa que el clasificador ha clasificado a una muestra como positiva y en la realidad es positiva.

- **Falso Positivo *(False Positive) (FP)*.** Representa aquella muestra que habiendo sido clasificada por el clasificador como positiva, es positiva.

- **Falso Negativo *(False Negative) (FN)*.** Indica que el clasificador ha clasificado a una muestra como positiva y en la realidad es negativa.

- **Verdadero Negativo *(True Negative) (TN)*.** Es la clase de una muestra que ha sido clasificada como negativa y en la realidad es positiva.

En el ámbito de la detección, que una muestra haya sido clasificada como positiva indica que el detector ha encontrado en ella al objeto buscado, si no, será clasificada como negativa. Si el detector acierta o falla es entonces cuando se le pone el adjetivo de verdadero/falso.

Con esta nueva manera de entender la clasificación de muestras, una forma representar el rendimiento del clasificador puede ser contando el número de muestras que han sido clasificadas dentro de cada clase de la realidad y presentarlo en una tabla. Esta forma de representación es lo que se llama como matriz de confusión:

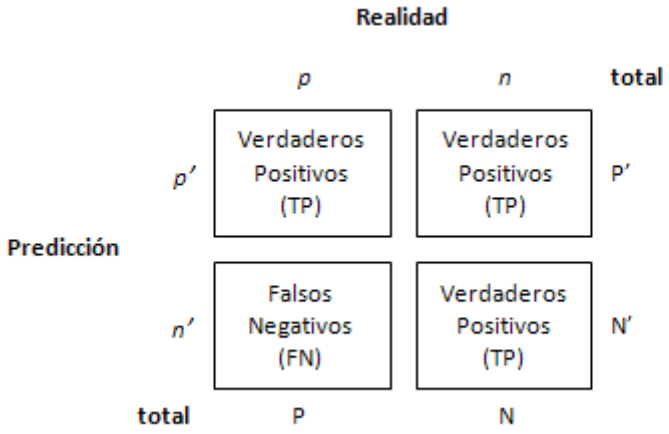


Figura 2.26: *Matriz de confusión.*

Sin embargo, la matriz de confusión por sí sola no aporta mucha información más allá del número de muestras de cada clase y éste no es un dato que se pueda utilizar para comparar con otros clasificadores. Pero ésta no es la finalidad de la matriz de confusión. La principal razón de construir la matriz de confusión es que a partir de ella se pueden deducir varios indicadores que permiten medir el rendimiento del clasificador. Estos indicadores surgen de relacionar el número muestras de cada clase entre ellas.

En el cuadro de la ***Figura 2.27*** se pueden ver los indicadores más importantes:

True Positive Rate (Sensitivity or Recall)	False Positive Rate (Fall-out)	True Negative Rate (Especifity)
$TPR = \frac{TP}{P} = \frac{TP}{TP + FN}$	$FPR = \frac{FP}{N} = \frac{FP}{FP + TN}$	$TNR = \frac{TN}{N} = \frac{TN}{FP + TN}$
Positive Predictive Value (Precision)	**Negative Predictive Value**	**Accuracy**
$PPV = \frac{TP}{TP + FP}$	$NPV = \frac{TN}{FN + TN}$	$ACC = \frac{P}{T} = \frac{TP + TN}{P + N}$

Figura 2.27: *Indicadores de rendimiento de un clasificador.*

Cada uno de estos indicadores de rendimiento mide una característica del clasificador. A continuación se explicará lo que representan cada uno de ellos:

- **Tasa de Verdaderos Positivos *(True Positive Rate) (TPR)*.** También conocida como sensibilidad *(sensibility)* o índice de recuerdo *(recall)*. Mide la capacidad del clasificador para clasificar correctamente las muestras positivas.

- **Tasa de Falsos Positivos *(False Positive Rate) (FPR)*.** Se conoce también como índice de olvido *(fall-out)*. Es un indicativo de la proporción de muestras negativas que el clasificador no es capaz de clasificar correctamente.

- **Tasa de Verdaderos Negativos *(True Negative Rate) (TNR)*.** Conocida además como especificidad *(specifity)*. Determina la capacidad del clasificador para clasificar adecuadamente muestras negativas.

- **Valor Predictivo Positivo *(Positive Predictive Value) (PPV)*.** Se corresponde con la precisión *(precision)*. Es una característica del clasificador que indica de las muestras clasificadas como positivas cuales los son realmente.

- **Valor Predictivo Negativo *(Negative Predictive Value) (NPV)*.** Representa la capacidad del clasificador para que las muestras clasificadas como negativas así lo sean.

- **Exactitud *(Accuracy) (ACC)***. Indica la capacidad para clasificar correctamente todas las muestras procesadas por el clasificador.

Dependiendo de los requisitos de la aplicación donde se implemente el clasificador, se tendrá que evaluar un indicador u otro. Al final se tendrá que elegir el clasificador que mejores indicadores presente.

2.4.3. Curva *ROC*

La **curva *ROC (Receiver Operating Characteristic)* [14] [17]** es una forma gráfica de representar el rendimiento de un clasificador binario y consiste en dibujar la tasa de falsos positivos *(FPR)* frente a la tasa de verdaderos positivos *(TPR)* para distintos valores de un umbral de discriminación. El análisis de la curva *ROC* se interpreta directamente como el beneficio/coste de realizar una clasificación.

Con lo visto hasta ahora solo se podría ser capaz de dibujar un punto de la curva *ROC*, puesto que solo se conoce un único valor de las tasas *TPR* y *FPR*, que es aquel que se calcula de la matriz de confusión teniendo en cuenta todas las muestras de evaluación. Para poder conseguir varios puntos de la curva, lo que se hace es definir un umbral, llamado de discriminación, que va seleccionando de forma incremental las muestras que son evaluadas hasta llegar al total de muestras. El umbral de discriminación actúa sobre una variable de la clasificación que generalmente es la puntuación con que ha sido clasificada una muestra. Esta puntuación representa la probabilidad de que una muestra pertenezca a la clase clasificada y se puede obtener del propio clasificador. Ahora, para cada valor del umbral se tendrá una matriz de confusión a partir de la cual se calcule un nuevo valor para las tasas y, en consecuencia, un nuevo punto para la curva *ROC*.

El proceso de dibujo de la *curva ROC* comienza asignando en una gráfica el eje de coordenadas "x" a la tasa de falsos positivos *(FPR)* y el eje de coordenadas "y" a la tasa de verdaderos positivos *(TPR)*. Para cada posible valor del umbral de discriminación, se seleccionan las muestras de evaluación cuya puntuación esté por debajo del umbral, se construye la matriz de confusión para esas muestras, se calculan las tasas *FPR* y *TPR*, y se dibuja un punto con sus valores en la gráfica. Este proceso se repite hasta que el conjunto de muestras seleccionado se corresponda con el total.

Cada punto de la curva *ROC* se calcula mediante las siguientes expresiones (referencias **[20] [21]** y **[24]**):

$$FPR(i) = \frac{FP(i)}{N(n)} = \frac{FP(i)}{FP(n) + TN(n)} \tag{2.15}$$

$$TPR(i) = \frac{TP(i)}{P(n)} = \frac{TP(i)}{TP(n) + FN(n)} \tag{2.16}$$

donde i representa el conjunto de muestras cuya puntuación está por debajo del umbral; n el conjunto de muestras total; $FPR(i)$ y $TPR(i)$, las tasas de falsos y verdaderos positivos para el umbral dado; $FP(i)$ y $TP(i)$, el número de muestras clasificadas como falsos y verdaderos positivos para cada umbral; $N(n)$ y $P(n)$, el número de muestras negativas y positivas totales; y, $TN(n)$ y $FN(n)$, el número total de muestras clasificadas como verdaderos y falsos negativos.

En la siguiente imagen se muestran algunos ejemplos de curvas *ROC* de distintos clasificadores:

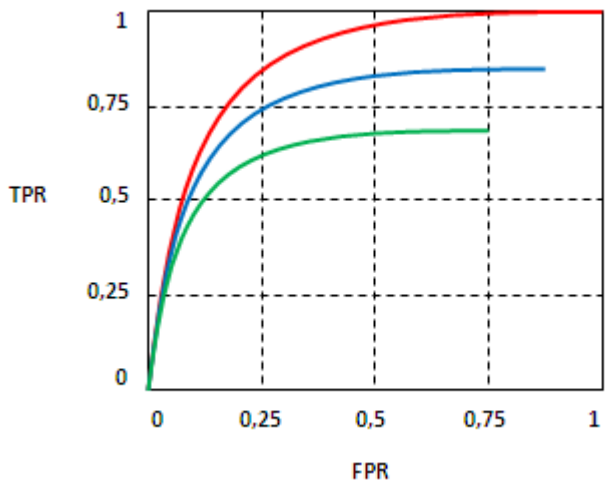


Figura 2.28: *Curva ROC.*

Una vez que se tiene dibujada la curva *ROC*, es momento de pasar a analizarla. La manera directa de hacerlo es viendo la forma de la curva que presenta. Así pues, para que el rendimiento de un clasificador se pueda considerar como aceptable es necesario que la curva *ROC* quede por encima de la diagonal que separa el espacio *ROC* en dos. Esta diagonal representa el límite que diferencia a una clasificación aleatoria de aquella que no lo es. Por tanto, cuanto más por encima de la diagonal se encuentre, mejor será el clasificador. Esto es fácil de interpretar ya que en ese caso predominará la tasa *TPR*, que es la que representa los aciertos, sobre la tasa *FPR*, que es la que representa los fallos.

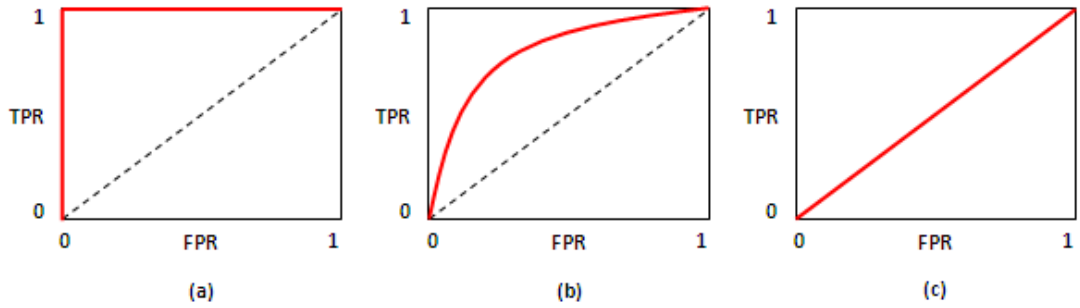


Figura 2.29: *Formas de la curva ROC. (a) Clasificador ideal. (b) Clasificador bueno. (c) Clasificador malo.*

Existen otros métodos que de forma numérica dan una estimación del rendimiento del clasificador a través de la *curva ROC*. De éstos, el método más utilizado es el **Área Bajo la Curva *(AUC) (Area Under Curve)* [20] [21] [24]**. Como su propio nombre indica, consiste en calcular el área que hay bajo la curva *ROC* (ver ***Figura 2.30***). Este indicador es otra manera, en este caso numérica, de comprobar cuanto predomina la tasa de verdaderos positivos *TPR* sobre la tasa de falsos positivos *FPR*.

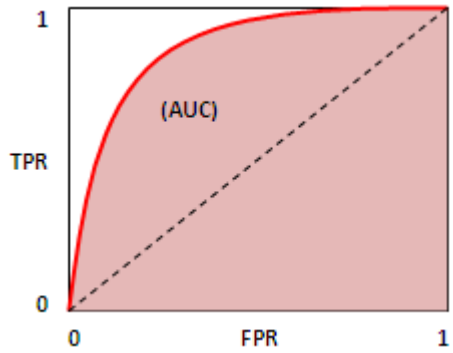


Figura 2.30: *Área bajo la curva (AUC).*

Conocidos los puntos de la curva *ROC*, el área bajo la curva AUC se puede calcular mediante interpolación rectangular como:

$$AUC = \sum_{k=0}^{n-1} TPR(k+1) \cdot \big(FPR(k+1) - FPR(k)\big) \tag{2.17}$$

donde k representa el índice de cada punto; n el número de puntos; y, TPR y FPR, las tasas de verdaderos y falsos positivos para cada punto. En este caso se considerará que el clasificador es aceptable si el indicador *AUC* tiene un valor por encima de 0,5. Cuanto mayor sea este valor mejor será el clasificador, siendo el caso ideal para aquel clasificador con un *AUC* de 1.

Otro método para valorar de forma numérica la curva *ROC* es la **Tasa de Error Igual *(EER) (Equal Error Rate)* [20] [21] [24]**. Este indicador de rendimiento representa la tasa de error cuando se tiene la misma probabilidad de clasificar incorrectamente las muestras positivas que las muestras negativas. Se calcula como el punto de intersección entre la *curva ROC* y la diagonal invertida del espacio *ROC*. En la siguiente figura se muestra la forma de calcularlo.

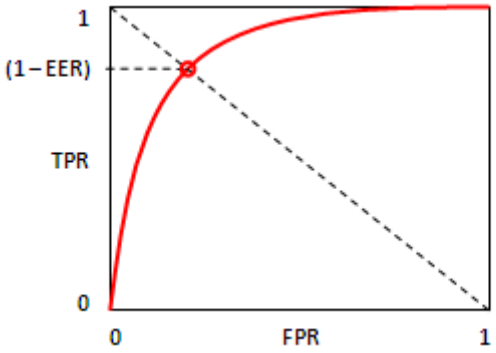


Figura 2.31: *Tasa de error igual (EER).*

En este caso, cuanto menor sea el valor de la tasa *EER*, mejor rendimiento presentará el clasificador. Hay que darse cuenta que sí la tasa *EER* es 0, la curva *ROC* obligatoriamente tiene que pasar por el punto de máximo rendimiento que es el (0, 1).

2.4.4. Curva *DET*

Una alternativa de la curva *ROC* es la **curva *DET (Detection Error Tradeoff)* [15] [18]**. En este caso, se representa la tasa de falsos negativos *(FNR)* junto con la tasa de falsos positivos *(FPR)* según va variando el umbral de discriminación. Esta curva surge para solucionar uno de los inconvenientes que presenta la curva *ROC*, que es que no tiene en cuenta el error de clasificar mal las muestras positivas. Esto se consigue analizando la tasa de falsos negativos *(FNR)*. El error de clasificar las muestras negativas viene dado por la tasa de falsos positivos *(FRP)*. En conclusión, la curva *DET* se puede interpretar como una medida del coste de clasificar erróneamente tanto las muestras positivas como las muestras negativas.

Algunos ejemplos de curvas *DET* se muestran en la siguiente imagen:

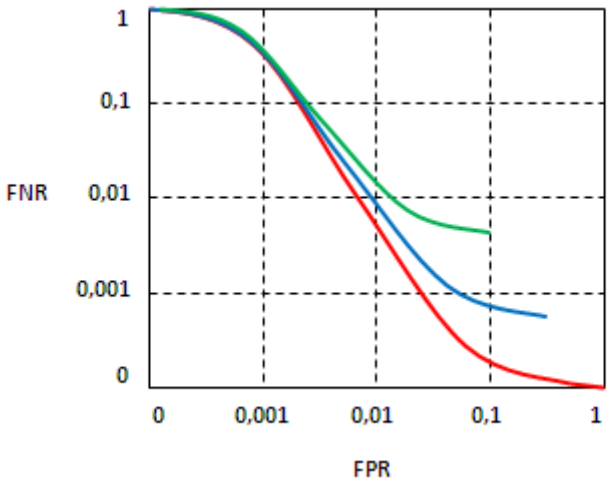


Figura 2.32: *Curva DET.*

El modo de calcular la curva *DET* es igual que el de la curva *ROC*, salvo que ahora sobre el eje "x" se representa la tasa de falsos negativos *(FNR)* y sobre el eje "y" la tasa de falsos positivos *(FPR)*. Para resaltar más la forma de la curva *DET*, los ejes se suelen representar en escala logarítmica. Los puntos de la curva *DET* se calcula con las siguientes expresiones (referencias **[20] [22] [25]**):

$$FPR(i) = \frac{FP(i)}{N(n)} = \frac{FP(i)}{FP(n) + TN(n)} \tag{2.18}$$

$$FNR(i) = \frac{FN(i)}{P(n)} = \frac{FN(i)}{TP(n) + FN(n)} \tag{2.19}$$

donde i representa el conjunto de muestras cuya puntuación está por debajo del umbral; n el conjunto de muestras total; $FPR(i)$ y $FNR(i)$, las tasas de falsos positivos y negativos para el umbral dado; $FP(i)$ y $FN(i)$, el número de muestras clasificadas como falsos positivos y negativos para cada umbral; $N(n)$ y $P(n)$, el número de muestras negativas y positivas totales; y, $TN(n)$ y $FN(n)$, el número total de muestras clasificadas como verdaderos y falsos negativos.

A la hora de analizar de forma visual la curva *DET*, se tiene que buscar que pasé lo más cerca posible del punto de máximo rendimiento que, en este caso, se corresponde con el punto situado en (0, 0). Cuanto más cerca esté de ese punto, menor es el error de clasificación que comete el clasificador.

2.4.5. Curva *PR*

La **curva *PR (Precision and Recall)* [16] [19]** es otra forma gráfica de representar el rendimiento de un clasificador. Se obtiene de comparar, para distintos valores del umbral de discriminación, la tasa de verdaderos positivos *(TPR)*, llamada en este caso *recall (REC)*, frente al valor predictivo positivo *(PPV)*, también conocido como *precision (PRE)*. Esta curva lo que permite es analizar la capacidad del clasificador para mantener la precisión a la hora de clasificar correctamente muestras positivas según se va incrementando el número de muestras procesadas.

A continuación muestra algunas formas que puede adoptar la curva *PR*:

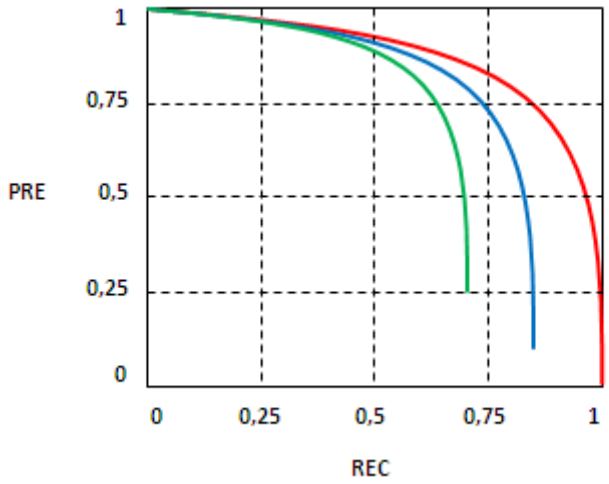


Figura 2.33: *Curva PR.*

El procedimiento a seguir para calcular la curva *PR* sigue siendo el mismo que para los dos tipos de curvas vistos anteriormente. La diferencia se encuentra en que ahora se representa en el eje "x" el *recall* y en el eje "y" la precisión. El valor de cada uno de los puntos de la curva se obtiene de aplicar, sobre el conjunto de muestras de evaluación y para los distintos valores del umbral de discriminación, las siguientes expresiones (referencias **[20] [23]** y **[26]**):

$$REC(i) = \frac{TP(i)}{P(n)} = \frac{TP(i)}{TP(n) + FN(n)} \tag{2.20}$$

$$PRE(i) = \frac{TP(i)}{P(i)} = \frac{TP(i)}{TP(i) + FN(i)} \tag{2.21}$$

donde i representa el conjunto de muestras cuya puntuación está por debajo del umbral; n, el conjunto de muestras total; $REC(i)$ y $PRE(i)$, el *recall* y la precisión para el umbral dado; $TP(i)$, el número de muestras clasificadas como verdaderos positivos para cada umbral; $P(n)$ y $P(i)$, el número de muestras positivas para el total de muestras y para el umbral dado; y, $FN(n)$ y $FN(i)$, el número de muestras clasificadas como falsos negativos para el total de muestras y el umbral dado.

Tras calcular y dibujar la curva *PR*, se puede extraer una primera evaluación de rendimiento analizando el aspecto que presenta la propia curva. Así pues, se considera que el clasificador tiene mayor y mejor precisión cuanto más elevada y horizontal se mantenga su curva *PR*. La altura de la curva es un indicativo de lo preciso que es el clasificador mientras que la horizontalidad representa lo constante que se mantiene la precisión con el aumento del número de muestras procesadas.

También se puede determinar la precisión del clasificador de forma numérica haciendo uso del área bajo la curva AUC. En este caso, se utiliza una interpolación trapezoidal para su cálculo **[20] [23] [26]**:

$$AUC = \frac{1}{2}\sum_{k=0}^{n-1}\big(PRE(k+1) + PRE(k)\big)\cdot\big(REC(k+1) - REC(k)\big) \tag{2.22}$$

donde k representa el índice de cada punto de la curva; n, el número de puntos; y, PRE y REC, la precisión y el recall de cada punto. La *AUC* será mayor cuanto mejor sea la precisión del clasificador.

Para este tipo de curva, la *AUC* no suele ser un indicador fiable. Esto es debido a que, generalmente, la curva *PR* presenta picos y variaciones muy rápidas que no tienen ninguna repercusión en el análisis del rendimiento pero que si alteran el valor de la *AUC*, pudiendo llegar a confundir su interpretación. Para evitar este inconveniente, se define un nuevo indicador, basado en la *AUC*, que consigue ignorar esos picos aplicando interpolación. A este indicador se le denomina como **Precisión Media Interpolada en 11 puntos *(Average Precision Interpolated in 11 points) (API11)*** **[20] [23] [26]** y consiste en calcular el promedio de la precisión interpolada sobre 11 puntos de la *curva PR* correspondientes a 11 niveles de recall equidistantes. La expresión para calcular el indicador $API11$ es la siguiente:

$$API11 = \frac{1}{11}\sum_{r} PRE_{interp}(r)$$
$$r \in \{0, 0.1, 0.2, \dots, 1\} \tag{2.23}$$

con PRE_{interp} la precisión interpolada sobre 11 puntos con *recall* r y calculada como:

$$PRE_{interp}(r) = \max_{\mathrm{REC(k)}\geq \mathrm{r}} PRE(k)$$
$$k \in \{0, 1, 2, \dots, n-1\} \tag{2.24}$$

donde k es el índice de los puntos de la curva; n, el número de puntos de la curva; y, PRE y REC, la precisión y el *recall* de cada punto.

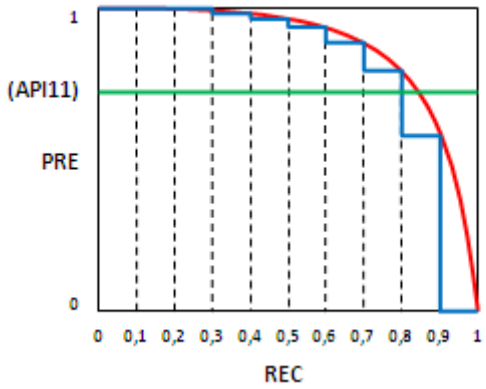


Figura 2.34: *Precisión media interpolada en 11 puntos (API11).*

De la ***Figura 2.34*** se puede intuir que cuanto mayor sea el valor del indicador *API11* mayor precisión presentará el clasificador evaluado.

Capítulo 3

Desarrollo

3.1. Recopilación y Etiquetado de Imágenes de Muestra de Semáforos

3.1.1. Introducción

Como ya se comentó, la parte principal del detector está formada por un clasificador en cascada que irá evaluando sobre que ventanas de las imágenes de entrada se encuentran los semáforos. Para que el clasificador sea capaz de distinguir los semáforos del resto de objetos, es necesario someterle a un proceso de entrenamiento en el que, mediante un algoritmo de aprendizaje y un conjunto de muestras positivas (contienen al objeto) y negativas (no lo contienen), se vaya ajustando hasta obtener una determinada tasa de aciertos/fallos en la clasificación de dichas muestras.

Este mismo proceso de entrenamiento se encuentra implementado bajo una serie de aplicaciones que incorporan las librerías de *OpenCV* y que han sido las utilizadas para entrenar el clasificador de semáforos. Para poder realizar el entrenamiento, estas aplicaciones requieren justamente de una colección de imágenes de muestra tanto positivas como negativas pero además de unos archivos de etiquetado que indiquen la clase de las muestras. Esta es la razón por la que la primera tarea en el proceso de desarrollo del sistema detector de semáforos sea la **recopilación y etiquetado de imágenes de muestra**.

Así pues, lo que se describirá en este apartado son todos los pasos que se llevaron a cabo para obtener una colección de imágenes que permitiera entrenar al clasificador de semáforos con las herramientas que proporcionan las librerías de *OpenCV*. En el proceso se contará desde el método utilizado para recopilar las imágenes de muestra hasta como etiquetarlas con la ayuda de una aplicación diseñada para ello.

3.1.2. Recopilación de Imágenes de Muestra

3.1.2.1. Alternativas de Recopilación

En el proceso de recopilación de imágenes de muestra de semáforos se encontraron varias alternativas que fueron siendo probadas hasta encontrar la mejor solución. A continuación se describen cada una de ellas:

- La primera alternativa que se pensó fue la búsqueda en internet de bases de datos con imágenes de semáforos que se pudieran descargar de forma libre. En este intento, si bien se hallaron bases de datos para un amplio abanico de objetos (vehículos, animales, personas, etc.), no se consiguió encontrar ninguna base de datos con semáforos, por lo que se decidió pensar en otra alternativa.

- La siguiente opción que se intentó fue descargar imágenes de semáforos una a una desde internet haciendo uso del buscador de imágenes de *Google*. Después de unas cuantas descargas se vio que este proceso era poco eficiente ya que conllevaba demasiado tiempo y el número de muestras requerido era bastante elevado.

- Otra propuesta fue la descarga de vídeos de internet que contuvieran secuencias del tráfico grabadas a bordo de un vehículo. Los videos se descompondrían en imágenes y se seleccionarían aquellas en las que se encontrasen semáforos. La gran ventaja de este método es que se consiguen obtener una gran cantidad de muestras, ya que se tienen tantas imágenes como fotogramas tenga el vídeo. Esta opción también quedo descartada ya que se comprobó que los vídeos descargados no tenía la suficiente calidad como para poder extraer los semáforos con una buena resolución. Aún así, sirvió como idea para la alternativa que posteriormente se adoptó.

- Finalmente, la opción elegida fue la de grabar videos del tráfico directamente desde el habitáculo de un vehículo utilizando una cámara acoplada al mismo. De esta manera, se consiguieron secuencias de video con buena calidad y donde los semáforos aparecen bien definidos. Además, esta opción permite centrar las grabaciones en las zonas de tráfico donde se encuentran los semáforos, evitando así revisar secuencias de video donde no haya ninguno. Una vez realizadas las grabaciones, los vídeos se convierten en imágenes utilizando algún programa informático y de estas se escogen aquellas en las que aparezcan semáforos.

3.1.2.2. Equipo de Grabación

Una vez que se ha reflexionado que la opción de grabar vídeos desde un vehículo mediante una cámara era la mejor de todas las opciones que se barajaron, el siguiente paso fue preparar todo el equipo necesario para llevar a cabo este plan. Hay que comentar que todo el equipo que se presenta a continuación es propiedad del autor del proyecto.

Lo primero que se debía plantear era que cámara de grabación de vídeo emplear. Dado que no se disponía de ningún otro recurso, se decidió utilizar la cámara de video de un teléfono móvil. Hoy en día, la calidad que presentan las cámaras de la mayoría de los teléfonos móviles del mercado permite el que se pueda emplearlos en aplicaciones como esta. En concreto el teléfono móvil al que se hace referencia es un modelo Iphone 4 de Apple, el cual, dispone de una cámara de video con grabación en calidad HD 720p a 30 fps **[27]**.

Figura 3.1: *Teléfono móvil Iphone 4.*

Para acoplar el teléfono móvil en el habitáculo del vehículo, se hizo uso de un soporte específico para ello. Este soporte dispone de un sistema de sujeción con ventosa que permite situar el teléfono móvil en cualquier punto del parabrisas. Con esto se consigue que el campo de visión de la cámara se enfoque en la vía emulando la propia vista del conductor. Además, dispone de un brazo orientable con el que se puede ajustar el ángulo de grabación de la cámara.

Figura 3.2: *Soporte del Iphone 4 para vehículos.*

Por último, se necesita de un vehículo donde se pueda acoplar el equipo de grabación y desde el que se capture el tráfico. Específicamente, el vehículo utilizado es un modelo Opel Astra del año 1997.

Figura 3.3: *Vehículo de grabación.*

3.1.2.3. Lugar de Grabación

Con todo el equipo reunido y preparado, era el momento de plantearse donde se iba a grabar. Era evidente que se necesitaba una ciudad relativamente grande donde se espera que el nivel de la infraestructura del tráfico sea alto y así se asegurara el encontrar un gran número de semáforos. Para facilitar la tarea de grabar, también se requería de una ciudad conocida, por la que ya se hubiese circulado y se supiese en que zonas o calles de la misma se esperaría encontrar semáforos. Estos motivos sumados a la cercanía al domicilio del autor del proyecto, fueron la razón por la que se eligió como objetivo la ciudad de Alcalá de Henares. Para ubicar un poco Alcalá de Henares en el mapa, se comentará que es una ciudad perteneciente a la Comunidad de Madrid (España), situada a tan solo 33,7 kilómetros de distancia (o 30 minutos de desplazamiento en coche) del centro de Madrid.

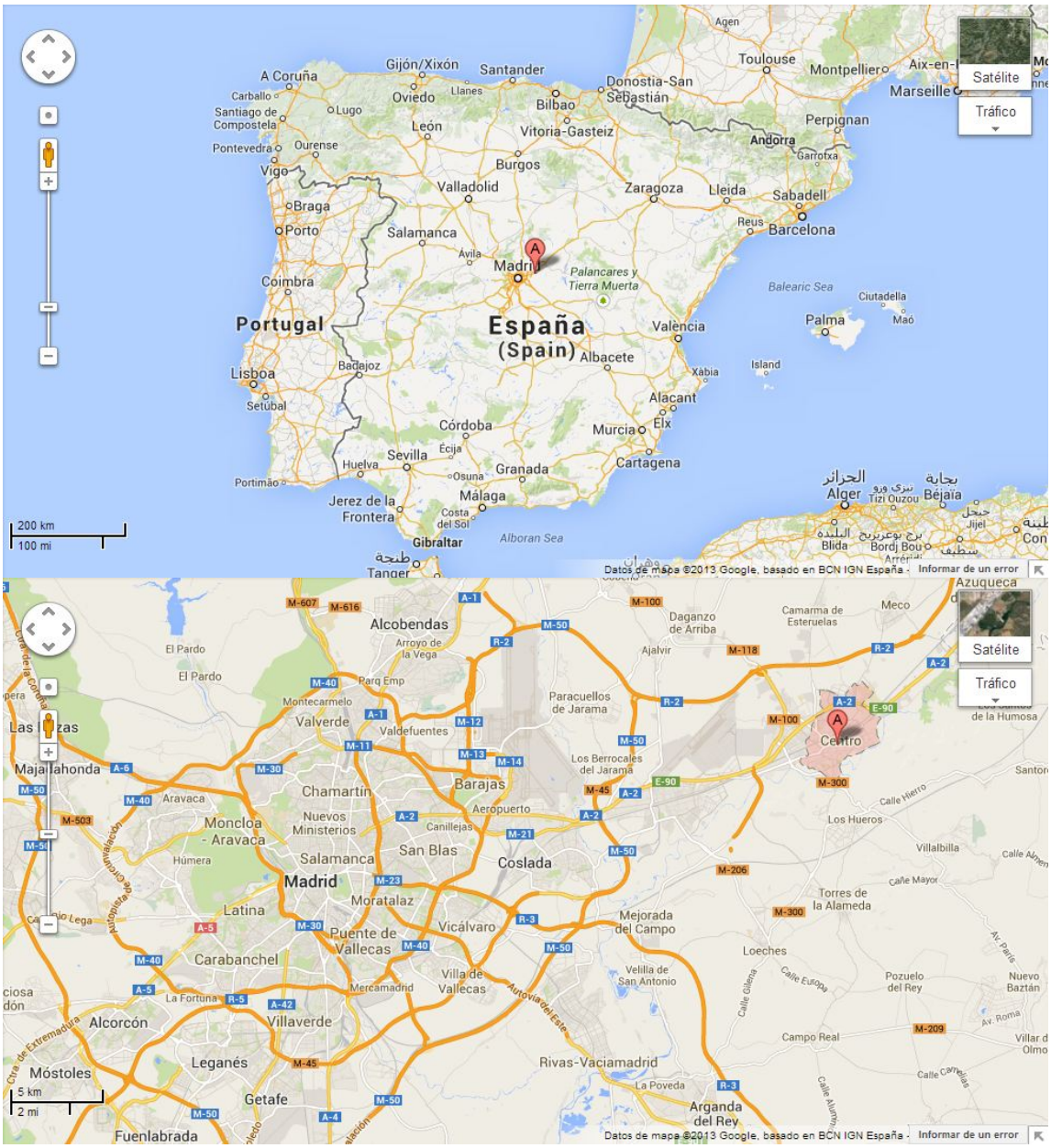


Figura 3.4: *Ubicación de Alcalá de Henares.*

3.1.2.4. Ruta de Grabación

Para que la grabación saliera lo mejor posible, se entendió que era clave planificar muy bien la ruta que se iba a seguir. Cuando se habla de una buena grabación lo que se pretende decir es que se capture el mayor número de semáforos en el menor tiempo posible. Con esto se consigue que en el vídeo predominen los fotogramas que tienen semáforos y no se pierda después mucho tiempo en clasificarlos.

Como ya se comentó anteriormente, la ciudad de Alcalá ya me era conocida por lo que resultó relativamente fácil estudiar un itinerario. El guión que se siguió para dibujar la ruta comenzaba por señalar aquellas calles en la que se sabía que había un gran número de semáforos. Después, se iban conectando una a una estas calles, generalmente por cercanía entre ellas, hasta conseguir formar trayectos lo más rectos posibles para evitar callejear. Finalmente, los trayectos se fueron combinando hasta encontrar un circuito cerrado que recorriese todos de la forma más simple que se pudiera. Además, el circuito se eligió para que las calles se recorriesen en los dos sentidos y por tanto se capturaran el doble de semáforos.

En la siguiente imagen se muestra la ruta finalmente elegida para grabar, dibujada sobre un plano urbano de la ciudad de Alcalá:

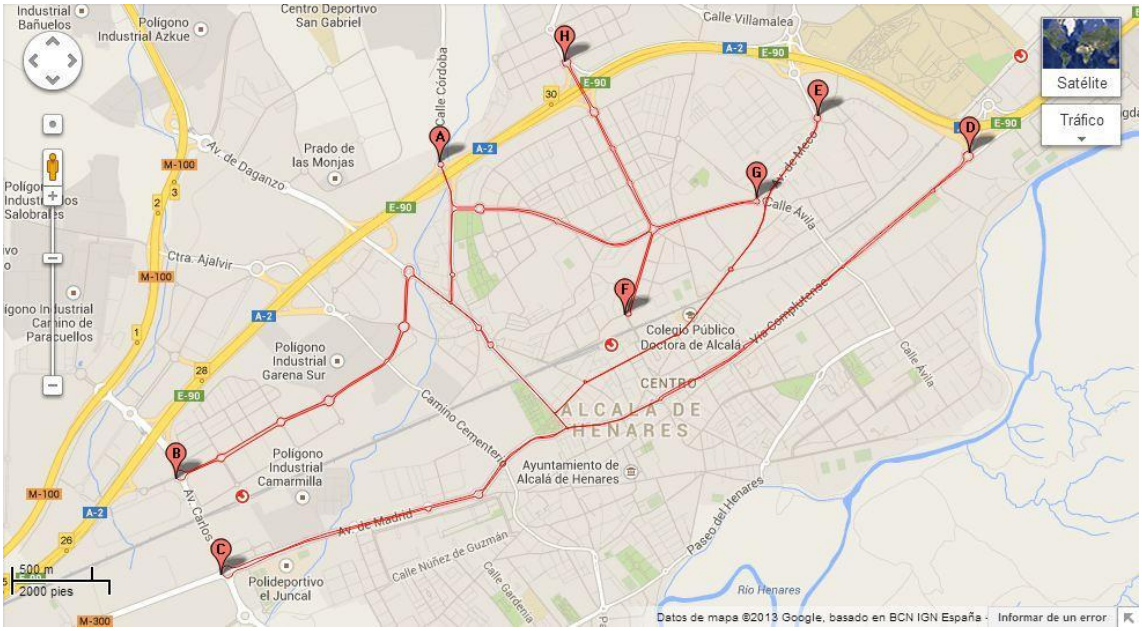


Figura 3.5: *Ruta de grabación sobre el plano urbano de Alcalá de Henares.*

Como se puede apreciar en el plano, la ruta es un circuito cerrado formado por 8 trayectos. Cada uno de estos trayectos conecta las calles situadas entre dos puntos ubicados en extremos de la ciudad. En total se tienen 8 puntos delimitadores que aparecen numerados en el mapa desde la letra A a la letra H. Darse cuenta que los trayectos son en su mayoría rectos y que las calles siempre se recorren en los dos sentidos.

3.1.2.5. Día de Grabación

Hasta este punto ya se tenía todo listo para grabar, solo quedaba una asunto pendiente por determinar que era elegir el día y la hora idóneos para realizar la grabación.

El nivel de tráfico era uno de los factores que podría determinar el día de grabación. Habría que evitar días y horas en las que no hubiese mucho tráfico para que no se entorpeciese y ralentizase la grabación. Teniendo en cuenta esto, se eligió un día festivo a una hora temprana de la mañana en la que se sabía que el nivel de tráfico sería bastante bajo.

Otro de los factores que podría influir en la elección del día de grabación era la climatología. No se podría grabar ni en los días lluviosos, porque las gotas que caen contra el parabrisas interferirían en las imágenes, ni en los días muy soleados, porque aparecerían reflejos y sombras tanto en el parabrisas como en los semáforos no pudiendo captarlos en buenas condiciones. Evidentemente tampoco se podría grabar de noche porque tan solo se captaría la luz del semáforo y no todo el módulo y ese no es el objetivo. Se comprobó que para que la grabación fuese perfecta tendría que realizarse en un día nublado y con mucha luz. Las nubes hacen que la luz se disperse consiguiendo lo que se llama una iluminación difusa, es decir, proveniente de todas las direcciones. La ventaja de la iluminación difusa es que evita las sombras y los reflejos comentados anteriormente.

3.1.2.6. Resultados de la Grabación

La grabación se llevo a cabo en aproximadamente 1 hora y media que fue lo que se tardó en recorrer la ruta planificada. En el proceso, se lograron capturar 23 vídeos (en formato *.mov*), los cuales hacen en total 1 hora y 7 segundos de grabación real (4,46 GB de espacio en disco). En este tiempo se pudieron contar un total de 411 semáforos capturados.

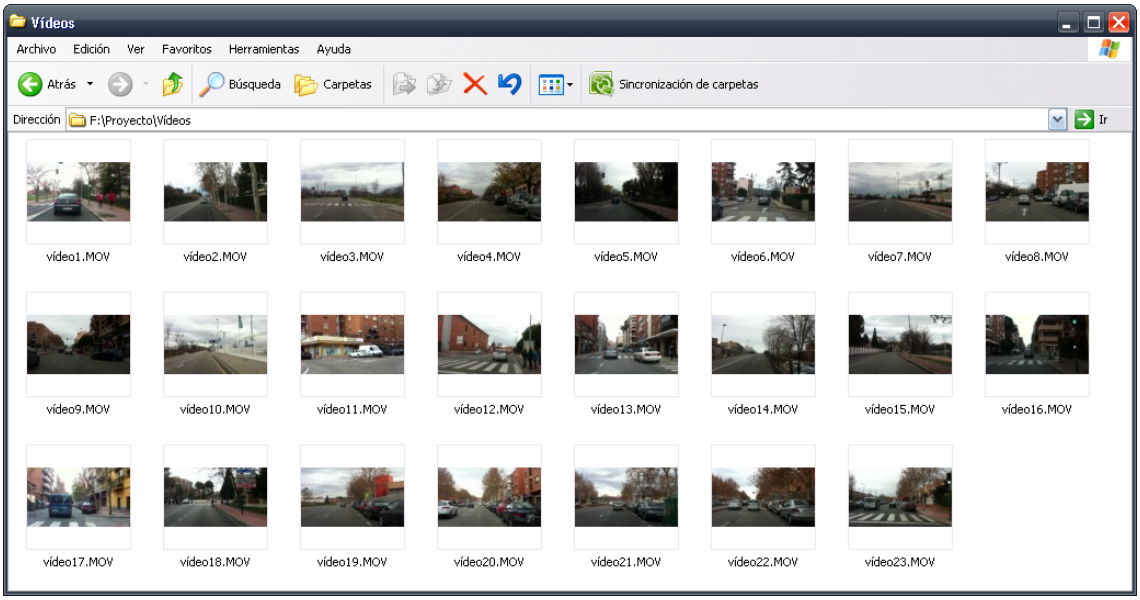


Figura 3.6: *Carpeta con los vídeos grabados.*

3.1.2.7. Convertir Vídeos en Imágenes

Ahora que se disponía de los archivos de vídeo, el siguiente paso fue extraer de ellos las imágenes. Para ello, se hizo uso de la aplicación *Marcador de Objetos*. Esta aplicación implementada en Visual Basic es un anexo al proyecto que fue desarrollada como trabajo dirigido para el Departamento de Ingeniería de Sistemas Automáticos de la Universidad Carlos III de Madrid. El objetivo para el que se diseñó esta aplicación era el agilizar todo el proceso de recopilación y etiquetado de muestras para el entrenamiento de clasificadores ya que es una tarea bastante tediosa.

Ejecutando el *Marcador de Objetos* y desplegando el menú *Herramientas* se puede encontrar una pequeña aplicación llamada *Convertidor de video a imágenes* ( ) que permite pasar los archivos de video grabados a simples archivos imagen.

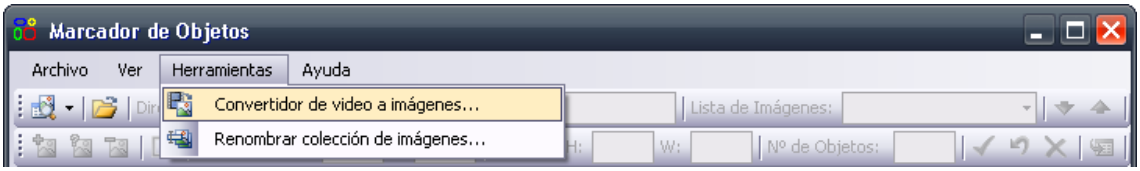


Figura 3.7: *Herramienta "Convertidor de Vídeo a Imágenes" de la aplicación "Marcador de Objetos".*

Una vez seleccionada la herramienta, aparece una ventana con las opciones que contiene y entre las que se encuentran: el nombre del archivo de video a convertir, el número de frames que tiene el video, el nombre del directorio donde se quieren guardar las imágenes y la relación conversión que indica cada cuantos frames del vídeo se genera una imagen.

Figura 3.8: *Ventana de la herramienta "Convertidor de Vídeo a Imágenes".*

Volviendo al caso que nos ocupa, uno a uno los vídeos grabados se fueron introduciendo en esta aplicación y se fueron obteniendo colecciones de imágenes con los fotogramas que se hubiesen seleccionado del video de acuerdo a la relación de conversión. Se eligió una relación de conversión de 5 frames/imagen ya que era la distancia a la que se encontraban diferencias perceptibles entre los fotogramas. Como los vídeos grabados trabajan a 30 frames por segundo, esto indica que se tiene 1 imagen por cada 166 milisegundos de video. En total se consiguieron obtener 20.888 imágenes (2,06 GB) en formato *.jpg* y de dimensiones 1280x720.

Aquí se muestra la carpeta con la colección de imágenes del primer vídeo:

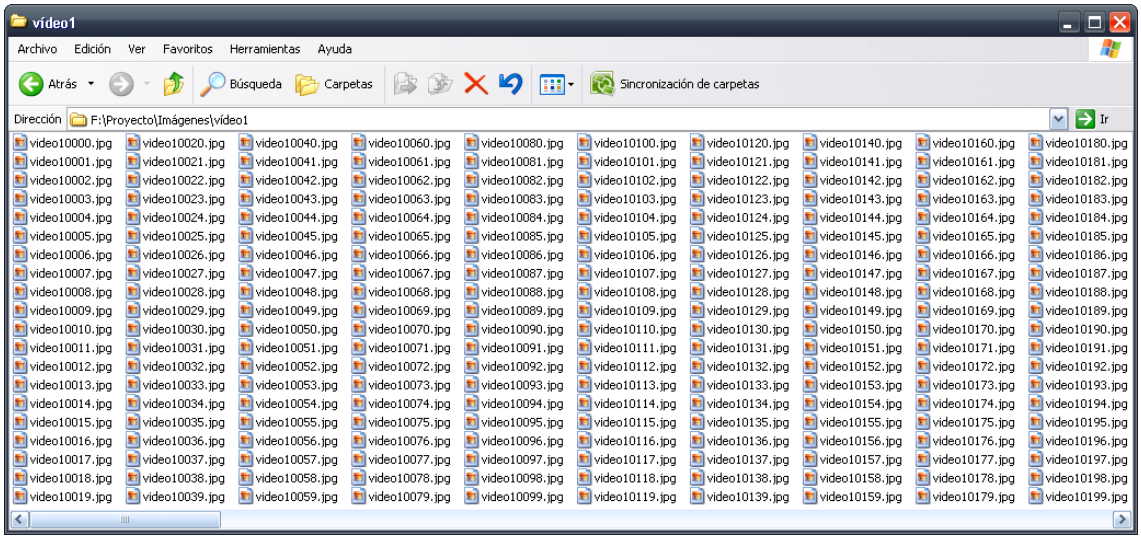


Figura 3.9: *Carpeta con las imágenes extraídas del primer vídeo.*

3.1.2.8. Renombrar Colección de Imágenes

Del paso anterior se tenía para cada video una carpeta con imágenes numeradas desde cero hasta el número total de imágenes de esa carpeta menos uno. Con el objetivo de facilitar el manejo de las imágenes en tareas posteriores, se pensó en juntar todas bajo una única carpeta. Pero para hacer esto sería necesario cambiar los rangos de numeración de tal forma que todas quedasen numeradas correlativamente. Como son muchas imágenes y sería imposible numerarlas una a una, la aplicación *Marcador de Objetos* también incorpora una herramienta que permite hacer este trabajo rápidamente.

Yendo a la aplicación *Marcador de Objetos* y seleccionando de nuevo el menú *Herramientas* se encuentra la opción *Renombrar colección de imágenes* (). Esta mini aplicación será la que se encargue de hacer el trabajo comentado antes.

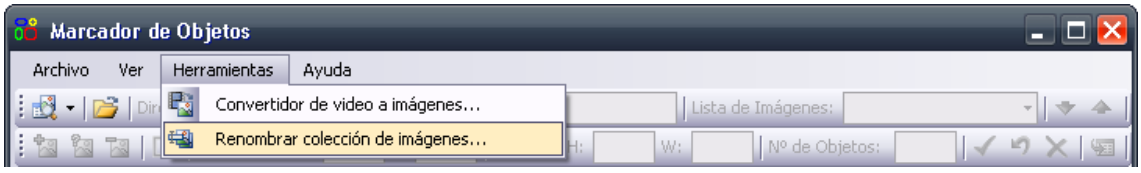


Figura 3.10: *Herramienta "Renombrar Colección de Imágenes" de la aplicación "Marcador de Objetos".*

Al clicar sobre el ítem *Renombrar colección de imágenes* se abre una ventana con las diferentes opciones de la herramienta que son: el directorio donde se encuentra la colección de imágenes, el número de imágenes que contiene ese directorio, el nuevo nombre que se le va a dar a las imágenes y, por último, el número desde el que comienza la numeración.

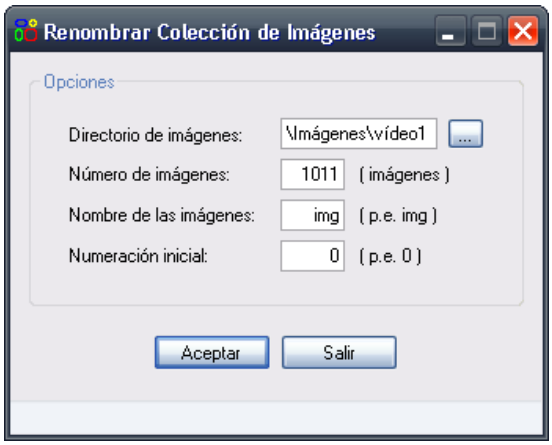


Figura 3.11: *Ventana de la herramienta "Renombrar Colección de Imágenes".*

Tras renombrar las imágenes del primer vídeo, el aspecto de la carpeta que las contiene quedó así:

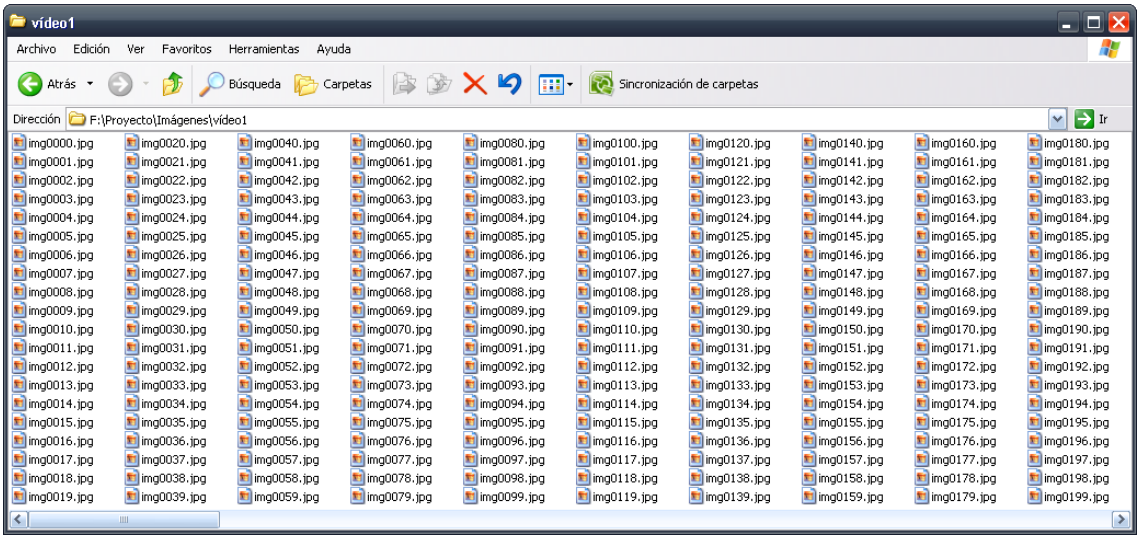


Figura 3.12: *Carpeta con las imágenes del primer vídeo renombradas.*

Este proceso se ha explicado para la primera colección de imágenes pero se fue realizando con cada una de las colecciones que existen para cada vídeo, de tal forma que la numeración de la siguiente colección de imágenes comenzase a partir de la numeración de la anterior. Una vez que se habían numerado correctamente todas las imágenes, se juntaron bajo una misma carpeta.

3.1.3. Etiquetado de Imágenes de Muestra

3.1.3.1. Introducción

Como ya se comentó en la introducción, el clasificador que incorpora el detector de semáforos ha sido construido y entrenado mediante un conjunto de herramientas auxiliares que incorporan las librerías de *OpenCV*. En concreto, estas herramientas son dos aplicaciones ejecutables que se pueden encontrar en dichas librerías bajo el nombre de *opencv_createsamples* y *opencv_traincascade* **[31]**. En otro apartado se explicará detalladamente el funcionamiento de estas dos aplicaciones. Por ahora, basta con saber que trabajan directamente sobre un conjunto de imágenes que contienen al objeto a detectar (positivas) y otro que no lo contienen (negativas). Para ello, requieren que se les introduzca como parámetro unos archivos en formato *.txt* que contienen información sobre las imágenes. A estos archivos se les ha dado en el proyecto el nombre de *listas*. Más adelante se entenderá porque. Como se requieren dos grupos de imágenes, también se necesitarán dos listas, una para las imágenes positivas y otra para las imágenes negativas.

3.1.3.2. Lista de Imágenes Positivas

La lista de imágenes positivas es utilizada por la aplicación *opencv_createsamples* para extraer de las imágenes positivas la ventana que contiene al objeto a detectar. Según se explica en la documentación, este archivo debe contener para cada imagen positiva la siguiente información:

- En nombre del archivo imagen incluida la ruta relativa al directorio donde se encuentre la aplicación *opencv_createsamples* (`FileName`).

- El número de objetos que contiene la imagen (`N`).

- La localización, coordenada "x" (`X`) y coordenada "y" (`Y`), y las dimensiones, ancho (`W`) y alto (`H`), del marco que contiene a cada objeto.

```
FileName  N  X1  Y1  W1  H1  X2  Y2  W2  H2  …  XN  YN  WN  HN
```

Figura 3.13: *Formato de un archivo lista de imágenes positivas.*

En el siguiente cuadro se muestra un ejemplo del contenido de este archivo lista:

```
../Imágenes/img_00.jpg  1  584  250  23  34
../Imágenes/img_01.jpg  1  736  249  35  77
../Imágenes/img_03.jpg  2  566  257  26  41  679  347  19  35
../Imágenes/img_05.jpg  2  461  194  27  50  628  329  19  43
../Imágenes/img_08.jpg  3  414  293  19  34  317  368  16  31  509  364  14  31
../Imágenes/img_09.jpg  3  500  310  24  38  598  379  20  29  419  376  21  29
```

Figura 3.14: *Ejemplo de un archivo lista de imágenes positivas.*

3.1.3.3. Lista de Imágenes Negativas

Por su parte, la aplicación *opencv_traincascade* es la que utiliza la lista de imágenes negativas, junto con otro archivo generado por *opencv_createsamples* y que contiene las muestras positivas, para construir y entrenar al clasificador. En este caso, lo único que debe contener el archivo lista es la ruta y el nombre de archivo de cada una de las imágenes negativas que se tenga. La ruta se escribe relativa al directorio donde esté ubicada la aplicación *opencv_traincascade*.

En el siguiente cuadro se muestra el contenido que podría tener un archivo lista de imágenes negativas:

```
../Imágenes/img_02.jpg
../Imágenes/img_04.jpg
../Imágenes/img_06.jpg
../Imágenes/img_07.jpg
../Imágenes/img_10.jpg
../Imágenes/img_12.jpg
```

Figura 3.15: *Ejemplo de un archivo lista de imágenes negativas.*

3.1.3.4. Etiquetado de Imágenes de Muestra

Como ya se podrá intuir, este trabajo de crear los archivos lista, también conocido como etiquetado de imágenes, es una tarea bastante laboriosa sobre todo cuando se sabe que son bastantes las imágenes necesarias para entrenar al clasificador. Esta es la razón por la que se diseñó la aplicación "Marcador de Objetos". Esta aplicación ya fue presentada en el proceso de recopilación de imágenes, pero en ese caso solo se utilizaron un par herramientas que fueron diseñadas en un segundo plano. El principal objetivo de esta aplicación era el de ofrecer un interfaz de usuario que permitiera realizar todo el proceso de etiquetado de imágenes de forma cómoda y rápida. De entre las funciones que ofrece la aplicación "Marcador de Objetos" las más importantes son:

- Abrir directorios con imágenes y mostrarlas en una ventana como si se tratase de un visor o galería de imágenes.

- Clasificar las imágenes por carpetas según el grupo al que pertenezcan: positivas, indeterminadas o negativas.

- Marcar sobre las imágenes las ventanas en la que se encuadran a los objetos de interés.

- Generar automáticamente los archivos lista, tanto para imágenes positivas como para imágenes negativas.

Para etiquetar una colección de imágenes mediante la aplicación *Marcador de Objetos* lo primero que hay que hacer es acceder a esta. Entonces aparecerá la ventana principal de la aplicación donde se distinguen una barra de menú, dos barras de herramientas y un espacio de trabajo.

Figura 3.16: *Ventana principal de la aplicación "Marcador de Objetos".*

Para abrir la colección de imágenes habrá que hacer clic directamente sobre el botón *Directorio de imágenes* ( ) o bien acceder desde *Archivo* → *Abrir* → *Directorio de imágenes*. En ese momento se abrirá un cuadro diálogo donde se pedirá que se seleccione la ubicación de la carpeta con las imágenes. Una vez seleccionada, se cargarán en la aplicación todas las imágenes y se mostrará la primera de ellas.

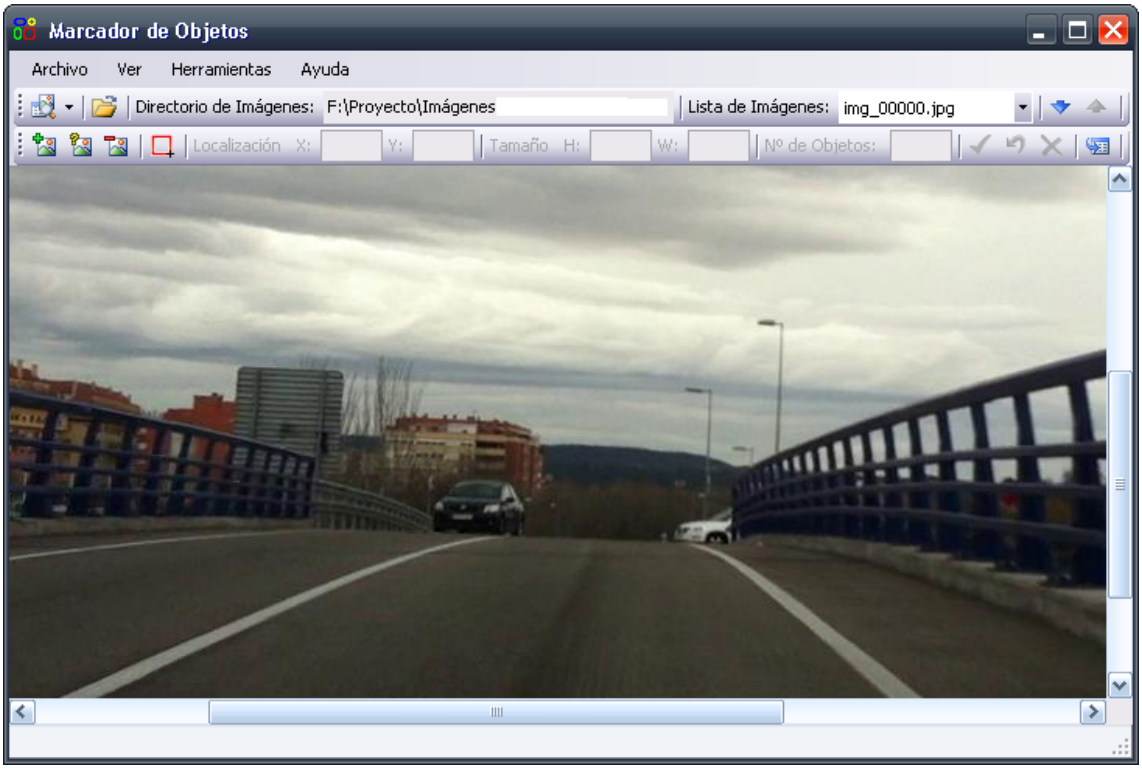


Figura 3.17: *"Marcador de Objetos" tras seleccionar el directorio de imágenes.*

Como se puede apreciar, ahora en la caja de texto *Directorio de Imágenes* aparece la ruta de la carpeta con la colección de imágenes seleccionada y en la otra caja texto *Lista de Imágenes* se muestra el nombre de archivo de la imagen que actualmente se está visualizando. Se puede ir cambiando la imagen mostrada o bien haciendo clic sobre el botón de *Lista de Imágenes* donde se desplegará una lista con los nombres de las imágenes y se podrá seleccionar una de ellas, o bien haciendo clic sobre los botones *Imagen Siguiente* e *Imagen Anterior*, o también desde teclado con las teclas (S) para siguiente y (W) para anterior.

Para clasificar la imagen mostrada dentro de los tipos de imágenes que hay (positiva, indeterminada y negativa), tan solo habrá que hacer clic sobre alguno de los botones que existen para ello. Si es una imagen positiva (aparece el objeto) se hará presionará el botón o la tecla (F), si es una imagen indeterminada (aparece el objeto pero no en buenas condiciones) se presionará el botón o la tecla (R) y si la imagen es negativa (no aparece el objeto) se presionará el botón o la tecla. Como se podrá comprobar, una vez que se han presionado estos botones, que en el directorio de las imágenes aparecen tres carpetas (Positivas, Indeterminadas y Negativas) en las que se irán copiando las imágenes según se vayan clasificando con la aplicación *Marcador de Objetos*.

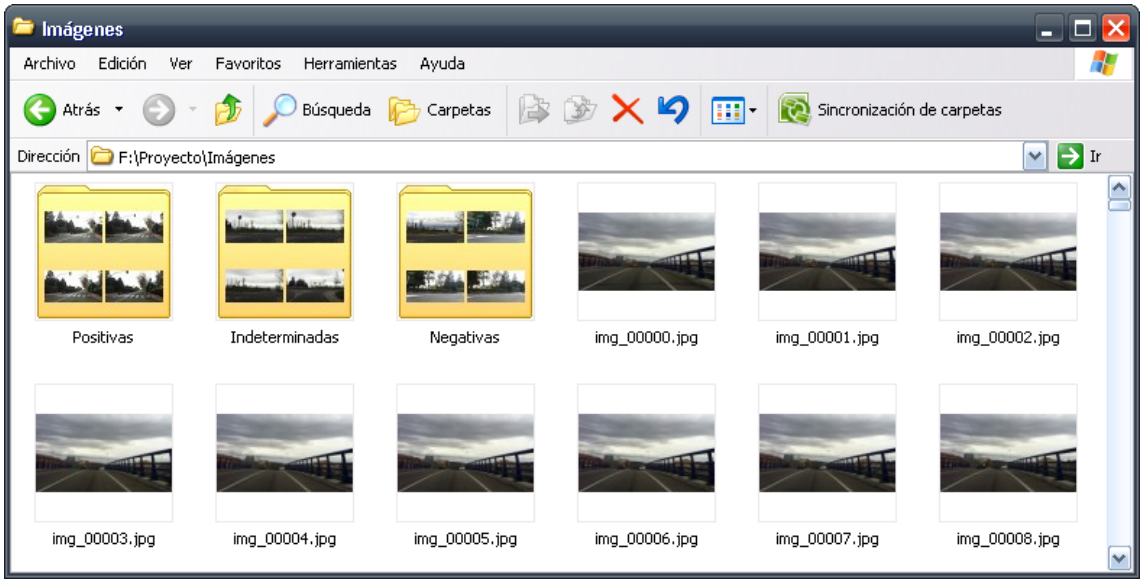


Figura 3.18: *Carpetas de clasificación de las imágenes.*

Si se quieren marcar los objetos que aparecen en la imagen mostrada se tendrá que entrar en el modo marcar haciendo clic sobre el botón *Marcar Objetos* o presionando la tecla (A). Para salir del modo marcar habrá que hacer clic otra vez sobre el mismo botón o presionando de nuevo la misma tecla. Una vez entrado en el modo marcar objetos, el cursor del ratón adquiere forma de cruz para indicar la localización del mismo sobre la imagen. Además, se dibujan unas líneas de referencia salientes del cursor para cuadrar bien horizontal y verticalmente a los objetos. Como se puede apreciar en la siguiente imagen, la localización del cursor se muestra en todo momento en las cajas de texto *Localización X* e *Y*. Para enmarcar al objeto, lo primero que hay que hacer es hacer clic sobre la posición donde se quiera situar la esquina superior izquierda del marco.

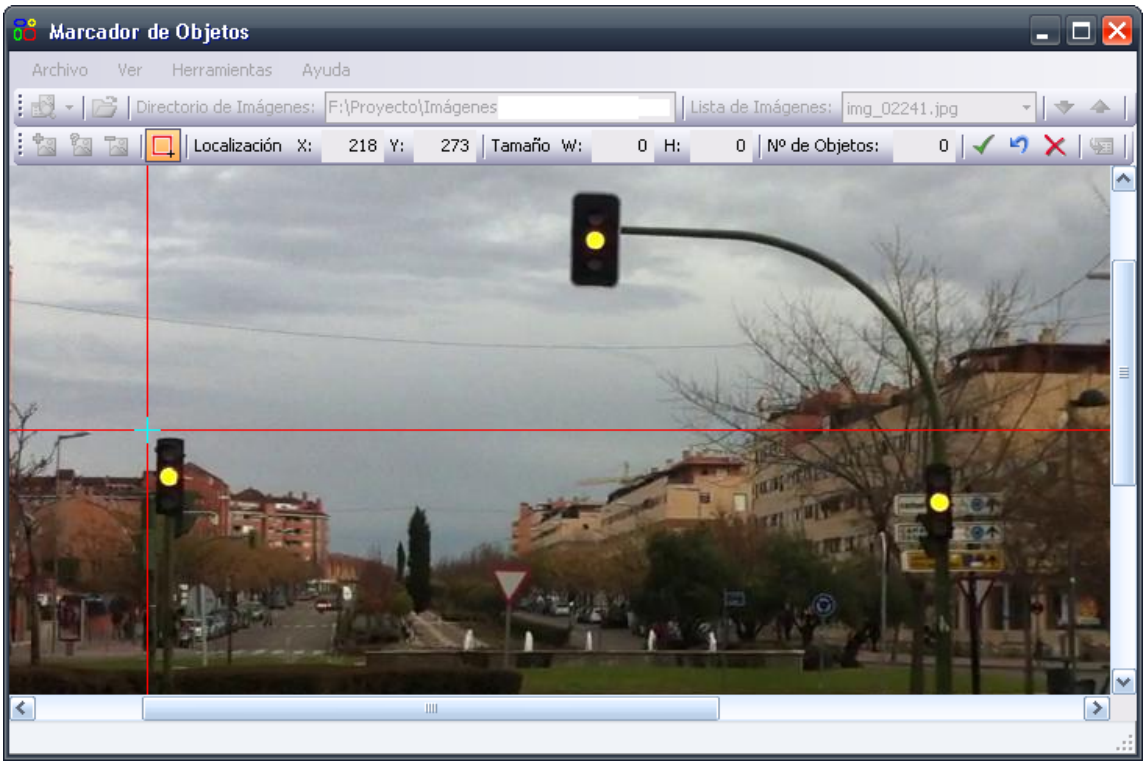


Figura 3.19: *Modo marcar objetos. Paso 1, clicar sobre la esquina superior izquierda.*

Después, se moverá el ratón hasta la esquina inferior derecha de lo que después será el marco. En el camino se podrá ir comprobando el tamaño del marco sobre las cajas de texto *Tamaño W* y *H*. Tras hacer clic en la segunda esquina, ya se tendrá el primer objeto marcado.

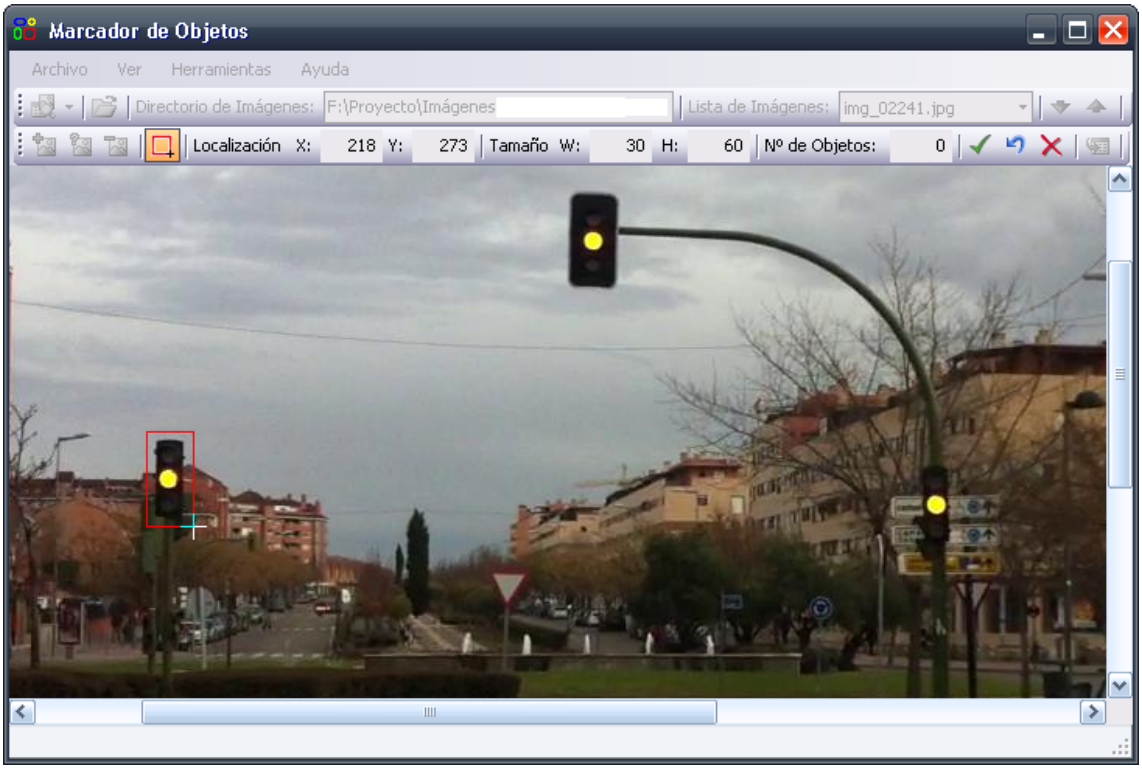


Figura 3.20: *Modo marcar objetos. Paso 2, clicar sobre la esquina inferior derecha.*

Ahora se seguirá el mismo proceso para marcar el resto de objetos. Se podrá comprobar el número de objetos marcados hasta el momento sobre la imagen en la caja de texto *Nº de Objetos*.

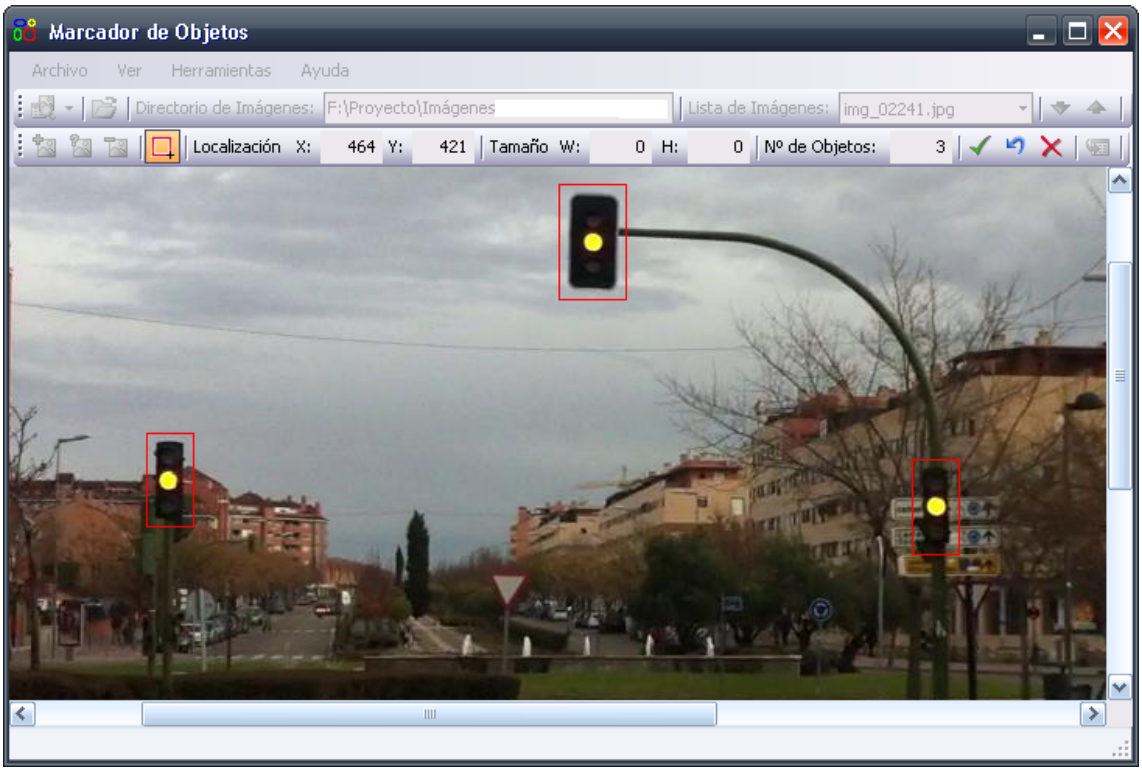


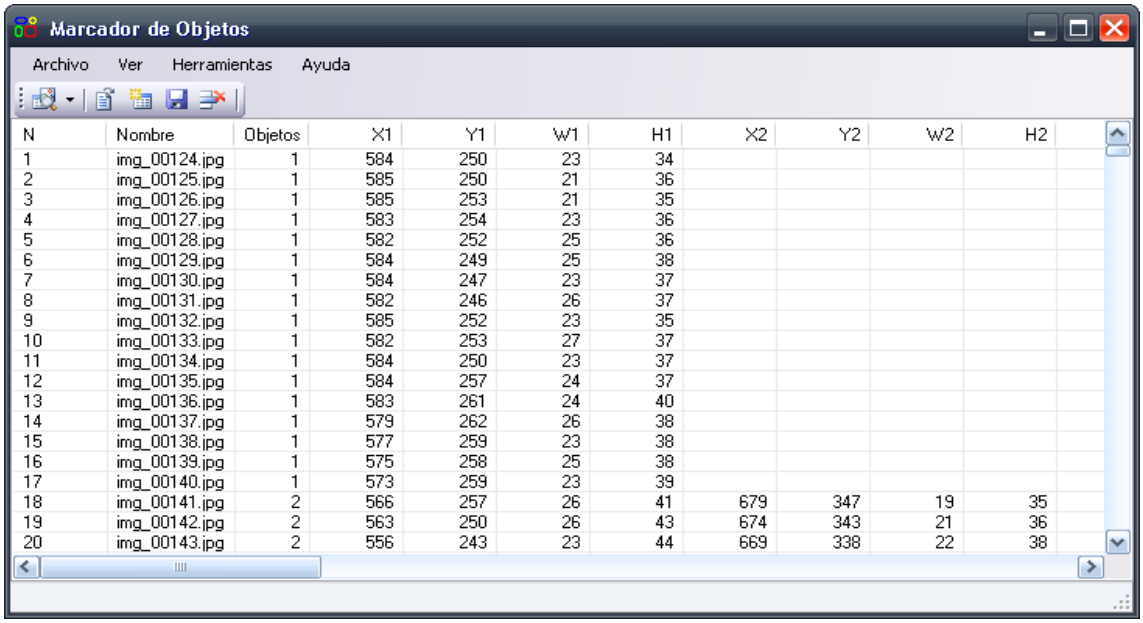
Figura 3.21: *Modo marcar objetos. Tras marcar todos los objetos.*

Una vez marcados los objetos y sin salir del modo marcar objetos, se podrán hacer tres acciones. La primera de ellas sería guardar las marcas y se realiza haciendo clic sobre el botón *Guardar marcas* (✓) o presionando la tecla (D). Otra acción que se podría realizar es borrar una a una las marcas y se podrá hacer con un clic sobre el botón *Deshacer marcas* (↶) o pulsando la tecla (E). Si por el contrario se quieren borrar todas las marcas, se hará clic sobre el botón *Eliminar marcas* (✗) o se pulsará la tecla (C).

Para salir del modo marcar objetos y pasar a una nueva imagen se pulsará otra vez el botón *Marcar objetos* (□) o se presionará la tecla (A).

Según se va avanzando en el proceso de marcar objetos, puede ser que se quieran revisar las marcas que se han ido guardando. Para hacer esto se tendrá que hacer clic sobre el botón desplegable *Cambiar vista* (🖹) y pinchar sobre el ítem *Lista* (☰), o también acceder desde el menú *Ver* → *Lista*. Si una vez accedido a la vista de la lista se quisiera retornar a la ventana de visualización de imágenes, no habría más que volver a abrir el desplegable del botón *Cambiar vista* (🖹) y en este caso pinchar sobre el ítem *Imagen* (🖼) o también desde el menú *Ver* → *Imágenes*.

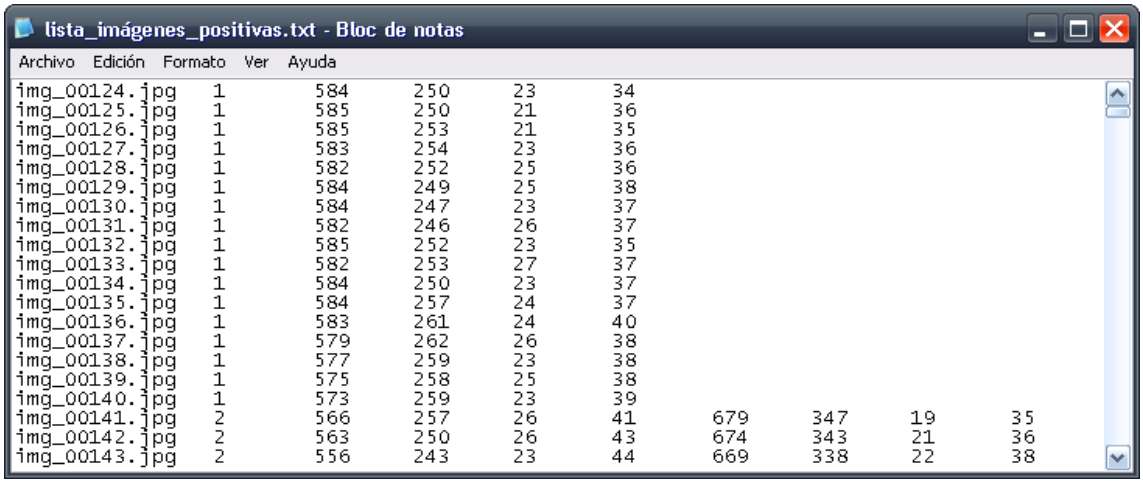
Tras entrar en la vista de lista de marcas, se podrá notar como la ventana principal de la aplicación cambia completamente. En este caso se ve además de la barra de menú, una única barra de herramientas y el listado con las marcas guardadas. Como se puede observar en la siguiente imagen, la lista guarda la información siguiendo el formato que ha de tener una lista de imágenes positivas.



N	Nombre	Objetos	X1	Y1	W1	H1	X2	Y2	W2	H2
1	img_00124.jpg	1	584	250	23	34				
2	img_00125.jpg	1	585	250	21	36				
3	img_00126.jpg	1	585	253	21	35				
4	img_00127.jpg	1	583	254	23	36				
5	img_00128.jpg	1	582	252	25	36				
6	img_00129.jpg	1	584	249	25	38				
7	img_00130.jpg	1	584	247	23	37				
8	img_00131.jpg	1	582	246	26	37				
9	img_00132.jpg	1	585	252	23	35				
10	img_00133.jpg	1	582	253	27	37				
11	img_00134.jpg	1	584	250	23	37				
12	img_00135.jpg	1	584	257	24	37				
13	img_00136.jpg	1	583	261	24	40				
14	img_00137.jpg	1	579	262	26	38				
15	img_00138.jpg	1	577	259	23	38				
16	img_00139.jpg	1	575	258	25	38				
17	img_00140.jpg	1	573	259	23	39				
18	img_00141.jpg	2	566	257	26	41	679	347	19	35
19	img_00142.jpg	2	563	250	26	43	674	343	21	36
20	img_00143.jpg	2	556	243	23	44	669	338	22	38

Figura 3.22: *Lista de imágenes positivas.*

De momento las marcas están solo guardadas en el espacio de trabajo de la aplicación por lo que si se cierra la aplicación se perdería toda la información. Para guardar la lista de marcas en un archivo *.txt*, se tendrá que pinchar sobre el botón *Guardar lista de imágenes* (). Esta acción también se podrá hacer desde el menú *Archivo* → *Guardar lista de imágenes* o *Archivo* → *Guardar lista de imágenes como*. Si la lista no se había guardado previamente aparecerá un cuadro de diálogo preguntado el directorio donde se quiere guardar y bajo que nombre. Abriendo el archivo lista una vez guardado se puede comprobar cómo se mantiene la información.




```
img_00124.jpg   1       584     250     23      34
img_00125.jpg   1       585     250     21      36
img_00126.jpg   1       585     253     21      35
img_00127.jpg   1       583     254     23      36
img_00128.jpg   1       582     252     25      36
img_00129.jpg   1       584     249     25      38
img_00130.jpg   1       584     247     23      37
img_00131.jpg   1       582     246     26      37
img_00132.jpg   1       585     252     23      35
img_00133.jpg   1       582     253     27      37
img_00134.jpg   1       584     250     23      37
img_00135.jpg   1       584     257     24      37
img_00136.jpg   1       583     261     24      40
img_00137.jpg   1       579     262     26      38
img_00138.jpg   1       577     259     23      38
img_00139.jpg   1       575     258     25      38
img_00140.jpg   1       573     259     23      39
img_00141.jpg   2       566     257     26      41      679     347     19      35
img_00142.jpg   2       563     250     26      43      674     343     21      36
img_00143.jpg   2       556     243     23      44      669     338     22      38
```

Figura 3.23: *Archivo de texto de una lista de imágenes positivas.*

Puede darse el caso de que se haya cometido algún error a la hora de marcar algunas de las imágenes y que se haya guardado en la lista. Para eliminar estas marcas erróneas de la lista se podrá seleccionar la fila o filas que las contienen y clicar sobre el botón *Eliminar filas* ( ).

La aplicación también permite cargar una lista que se hubiera guardado con anterioridad para continuar trabajando sobre ella. Entonces habrá que pinchar en el botón *Abrir lista de imágenes* ( ) y seleccionar sobre el cuadro de diálogo que aparece, el archivo de texto que contiene la lista a cargar. Esto también se podrá hacer accediendo a *Archivo → Abrir → Lista de imágenes*.

Otra posibilidad que ofrece la aplicación sería cargar una lista nueva y vacía sobrescribiendo la actual. Tan solo se tendrá que hacer clic sobre el botón *Nueva lista de imágenes* ( ). Si la lista actual no se ha guardado se pedirá que se guarde mediante un cuadro de diálogo.

Todo lo explicado hasta ahora sirve para crear una lista de imágenes positivas pero y si se quiere crear una lista de imágenes negativas. Pues bien, una vez que se hayan clasificado todas las imágenes de la colección, se tendrá una carpeta con el conjunto de imágenes negativas. Para generar la lista de imágenes negativas, en la vista de imágenes (recordar clic en *Cambiar vista* ( ) e *Imágenes* ( ), o *Ver → Imágenes*) tan solo se tendrá que abrir esta carpeta mediante el botón *Directorio de Imágenes* ( ) o *Archivo → Abrir → Directorio de imágenes* y una vez cargadas la imágenes, pinchar sobre el botón *Generar lista de imágenes negativas* ( ). Si se va ahora a la vista de lista de imágenes (recordar clic en *Cambiar vista* ( ) y *Lista* ( ), o *Ver → Lista*) se podrá comprobar cómo en este caso se ha generado una lista con el formato que ha de tener una lista de imágenes negativas. Para guardar la lista en un archivo de texto se hará clic en el botón de guardar ( ) o desde el menú *Archivo → Guardar lista de imágenes* o *Archivo → Guardar lista de imágenes como*.

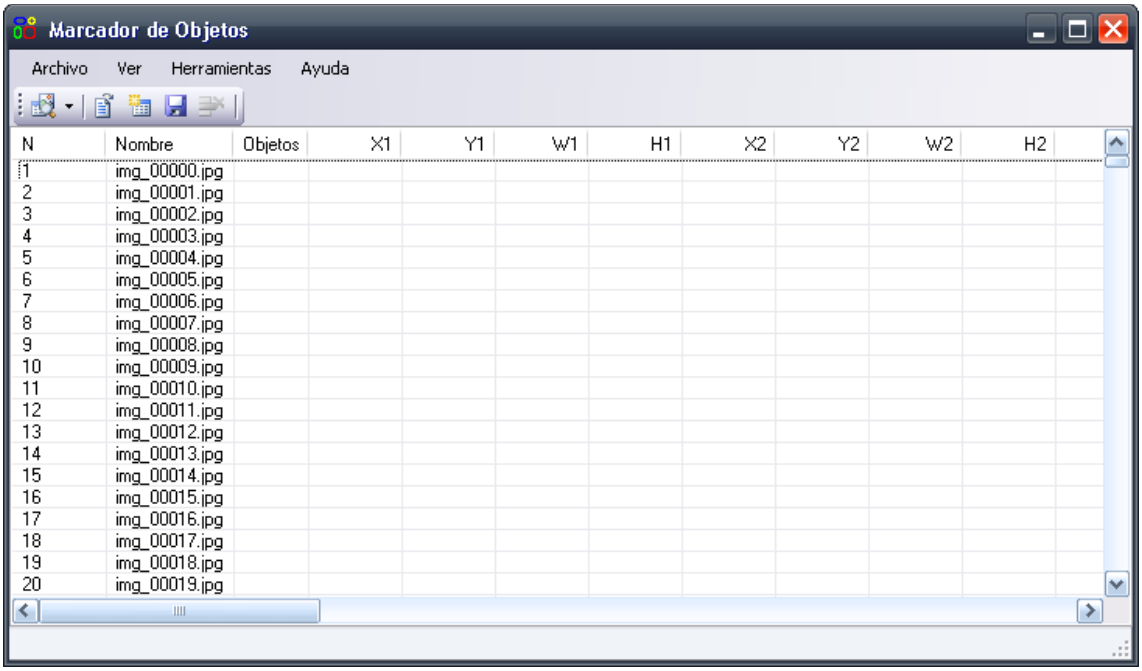


Figura 3.24: *Lista de imágenes negativas.*

Tras guardar el archivo lista de imágenes negativas, si se abre se puede comprobar cómo también para este caso se mantiene el formato y la información.

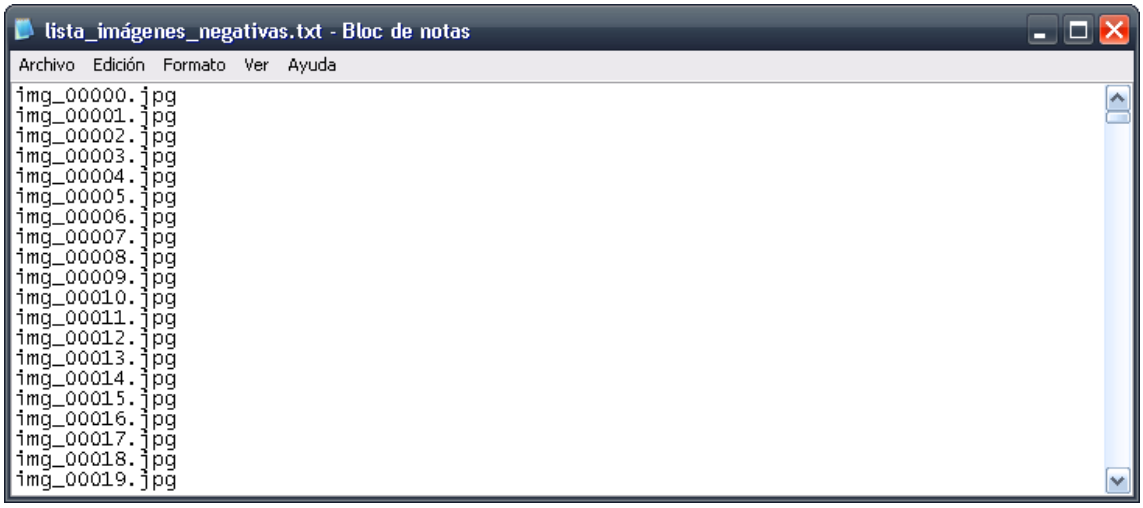


Figura 3.25: *Archivo de texto de una lista de imágenes negativas.*

Según se explicó en apartados anteriores, las aplicaciones de entrenamiento de las librerías *OpenCV* requieren que listas, tanto de imágenes positivas como de imágenes negativas, que tengan en el primer campo la ruta junto con el nombre del archivo imagen, donde la ruta es relativa al directorio donde estén las aplicaciones, pero la aplicación *Marcador de Objetos* solo escribe el nombre de archivo. Para añadir la ruta se puede recurrir a la herramienta *Remplazar* de cualquier aplicación procesadora de texto. Esto le otorga mayor flexibilidad al contenido de las listas si en algún momento se quiere cambiar la ubicación de las mismas o de las propias aplicaciones de entrenamiento.

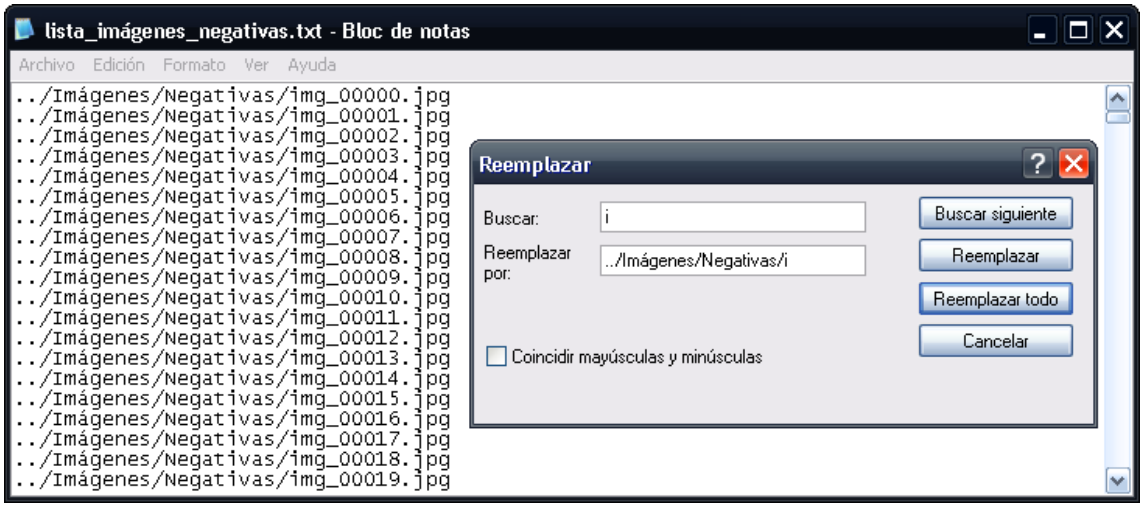


Figura 3.26: *Herramienta "Remplazar" de la aplicación "Bloc de notas".*

Aquí acabaría la explicación de cómo etiquetar una colección de imágenes haciendo uso de la aplicación *Marcador de Objetos*.

3.1.3.5. Resultados del Etiquetado

Retomando la tarea que nos ocupaba que era la de etiquetar el conjunto de imágenes que se habían recopilado del proceso de grabación, comentar que se realizó mediante la aplicación *Marcador de Objetos* de la misma manera que se ha descrito en el apartado de etiquetado. Aunque la aplicación agilizó mucho el trabajo, se tardó bastante tiempo (como dos o tres semanas en total) en etiquetar las 20.888 imágenes que se tenían. Al final de todo el proceso de etiquetado, se obtuvieron tres carpetas con las imágenes clasificadas en las que se contaron un total de 7.972 imágenes positivas (824 MB), 8.805 imágenes indeterminadas (873 MB) y 4.111 imágenes negativas (395 MB).

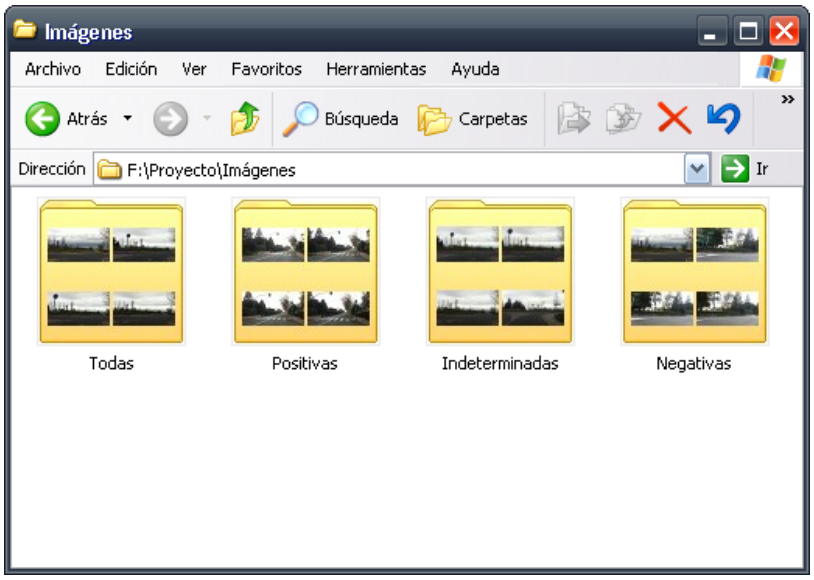


Figura 3.27: *Carpetas con la colección de imágenes clasificadas.*

También se obtuvieron los archivos de texto de etiquetado para las aplicaciones de entrenamiento de *OpenCV*, uno con la lista de imágenes positivas y otro con la lista de imágenes negativas. Dentro de las lista de imágenes positivas se llegaron a contar un total de 13.936 semáforos marcados.

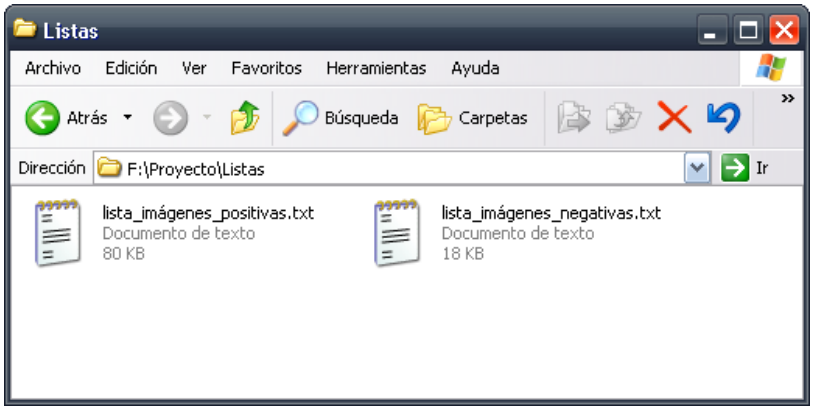


Figura 3.28: *Carpetas con la colección de imágenes clasificadas.*

3.2. Entrenamiento del Clasificador de Semáforos

3.2.1. Introducción

Una vez que se han recopilado las imágenes de muestra necesarias y se han clasificado según unos archivos de etiquetado, el siguiente paso en el desarrollo del sistema detector de semáforos será proceder con el **entrenamiento del clasificador**. De forma resumida, el entrenamiento del clasificador no es más que un algoritmo de aprendizaje que trabajará sobre el clasificador, ajustándolo hasta que sea capaz de clasificar las muestras de entrenamiento que se le pasen con un determinado error mínimo.

Como ya se fue introduciendo en apartados anteriores, el clasificador será implementado haciendo uso de una serie de herramientas que ponen a disposición para ello las librerías de *OpenCV*. Esta es la razón de que tanto las muestras como los etiquetados fueron adaptados para que cumplieran las especificaciones solicitadas por dichas herramientas. También se utilizaran otro grupo de herramientas que trabajan en conjunción con las de *OpenCV* y que han sido diseñadas específicamente para realizar algunas tareas que se consideran vinculadas al entrenamiento y que las otras no realizan.

Entonces, en este aparatado lo que se pretende describir es todo el proceso que se ha llevado cabo para obtener un clasificador de semáforos utilizando las herramientas de entrenamiento, tanto de *OpenCV* como las propias diseñadas, sobre las imágenes de muestra y archivos de etiquetado que se obtuvieron en el apartado anterior. Así pues, se comenzará describiendo el **equipo** sobre el que se ejecutará el entrenamiento. Después se presentarán las **aplicaciones** involucradas en el entrenamiento y como éstas se han organizado sobre un **directorio** para que trabajen en conjunción. Y finalmente, siguiendo la **secuencia de entrenamiento**, se describirá el proceso de ejecución de cada una estas aplicaciones hasta obtener el clasificador final.

3.2.2. Equipo de Entrenamiento

Para el entrenamiento del clasificador, no se utilizó el equipo personal que se había venido utilizando hasta ahora. El motivo está en que sí se quiere obtener un clasificador con un rendimiento lo suficientemente bueno como para emplearlo en aplicaciones reales, es necesario entrenarlo con una gran cantidad de imágenes de muestra y con unos parámetros de entrenamiento muy estrictos. Esto implica que el tiempo que se va a tardar en procesar todo ese volumen de muestras y en ajustar el clasificador hasta cumplir con esos requisitos tan justos, va ha ser bastante elevado. El entrenamiento podría llegar a durar incluso varias semanas en un equipo común.

Con el objetivo de agilizar el proceso de entrenamiento en busca de ahorrar tiempo, se utilizará en este caso un equipo con unos recursos hardware específicos para realizar este tipo de tareas rápidamente. Este equipo pertenece al Laboratorio de Sistemas Inteligentes del Departamento de Ingeniería de Sistemas Automáticos de la Universidad Carlos III de Madrid y se emplea para desarrollar proyectos relacionados con aplicaciones de visión artificial. Además de poseer unos recursos altos, este equipo tiene instalados un conjunto de complementos

para las librerías de *OpenCV* que permiten mejorar, más aún si cabe, el rendimiento de las aplicaciones que han sido desarrolladas con estas librerías. En la siguiente imagen, se muestra la ventana de detalles del equipo con los recursos que presentan el equipo:

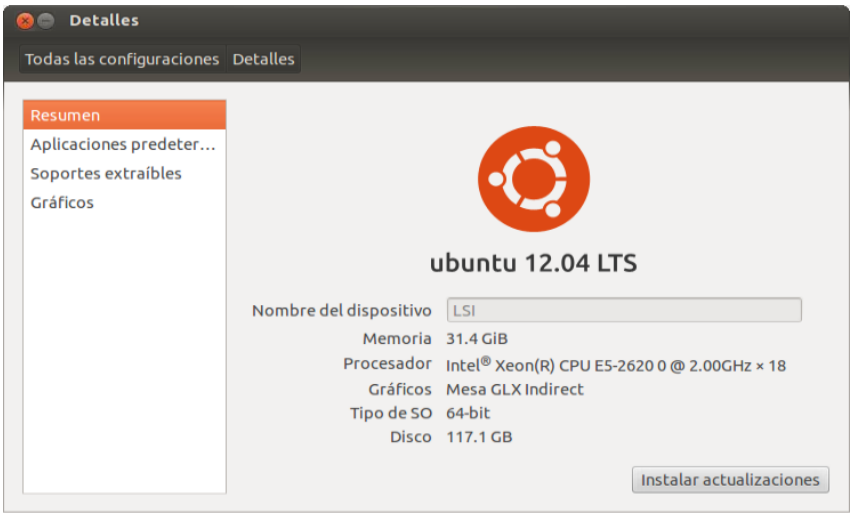


Figura 3.29: *Detalles del equipo de entrenamiento.*

De entre las prestaciones del equipo hay que destacar que:

- El equipo trabaja con el sistema operativo Linux Ubuntu versión 12.04 LTS **[28]**, muy común para el desarrollo y diseño de aplicaciones gracias a la libertad de trabajo que ofrece.

- Se dispone de una memoria RAM de 31.4 GiB (33.7 GB), que es bastante elevada si se compara con la de los equipos personales del mercado. Con esta memoria se conseguirán cargar aplicaciones muy pesadas como las del entrenamiento de clasificadores.

- En cuanto al procesador, es un modelo Intel Xeon E5-2620 **[29]** de 64 bits, con hasta 2.5 GHz de máxima velocidad de reloj y 16 MB de caché. Caben destacar los 6 núcleos y 12 hilos de ejecución de que dispone que le permitirán ejecutar hasta 12 tareas en paralelo.

- La tarjeta gráfica instalada en el equipo es una Nvidia modelo Tesla C2075 **[30]** con arquitectura CUDA diseñada para el procesamiento gráfico a gran velocidad. Muy útil en aplicaciones de visión por computador.

En resumen, las prestaciones que presenta este equipo permitirán que la carga de procesamiento que supone el entrenamiento de clasificadores sea más liviana y por tanto que se requiera de menos tiempo para ejecutarla.

3.2.3. Aplicaciones de Entrenamiento

Hasta ahora se ha hablado sobre unas aplicaciones que se utilizarán para realizar el proceso de entrenamiento del clasificador de semáforos, pero no se ha comentado ni cuales son ni que funciones tienen. Para ello, primero habrá que distinguir entre dos tipos de aplicaciones utilizadas: las que se han diseñado de forma propia y las que proporcionan las librerías de *OpenCV*. El motivo de diseñar nuevas aplicaciones además de las que ya se encuentran en *OpenCV* es que en éstas no se realizan algunas tareas que se consideran importantes para que todo el proceso de entrenamiento sea completo. Las aplicaciones de *OpenCV* se centran en la única tarea de entrenar al clasificador, y no tienen en cuenta otras tareas como pueden ser preprocesar las imágenes de muestra antes del entrenamiento o evaluar las características del clasificador entrenado. Concretamente estas son las dos nuevas funciones que implementan las aplicaciones diseñadas.

Más adelante se explicará cómo trabajar con todas estas aplicaciones pero por ahora se hará una breve descripción del papel que juegan. Siguiendo el orden de ejecución, las aplicaciones que participan en el entrenamiento del clasificador son las siguientes:

- ***preprocessimages***.

  Esta aplicación es una de las diseñadas de forma propia y se utiliza para preprocesar las imágenes de muestra antes de que sean utilizadas por las posteriores aplicaciones de entrenamiento. Las operaciones de preprocesamiento que realiza sobre las imágenes son configurables pero en total son 3: redimensionado, ecualizado y filtrado del ruido.

- ***opencv_createsamples***.

  Es una aplicación de las proporcionadas por *OpenCV* **[31]**. Su tarea es leer del archivo de etiquetado de imágenes positivas la ubicación de los objetos, extraer las ventanas donde se enmarcan y guardarlas en un archivo con formato especial *.vec*. Este archivo será el utilizado por la siguiente aplicación en lugar de las imágenes positivas.

- ***opencv_traincascade***.

  Esta aplicación está incluida también en *OpenCV* **[31]**. Se encarga de realizar realmente el entrenamiento del clasificador, es decir, es la aplicación que implementa el algoritmo de aprendizaje. Trabaja sobre el archivo *.vec* generado por *opencv_createsamples*, las imágenes negativas y el archivo de etiquetado también de imágenes negativas. Como resultado, genera un archivo en formato *.xml* con el clasificador entrenado.

- ***testcascade***.

  Esta aplicación ha sido diseñada específicamente para evaluar el rendimiento del clasificador entrenado en la aplicación anterior. Para ello necesita el archivo *.xml* que contiene al clasificador, unas imágenes positivas a poder ser diferentes a las utilizadas en el entrenamiento y un archivo de etiquetado con la ubicación de los objetos en esas imágenes. Proporciona un archivo de texto *.txt* con los indicadores y los puntos de las curvas de rendimiento.

3.2.4. Directorio de Entrenamiento

Cuando se estudió la manera de entrenar a los clasificadores con las aplicaciones de *OpenCV* y se comenzó a trabajar con ellas, se encontró que el proceso se volvía algo tedioso si se quería realizar más de un entrenamiento. Como estas herramientas trabajan sobre la línea de comandos, cada vez que se quería entrenar un nuevo clasificador había que estar abriendo el terminal, situar el directorio activo en el directorio de trabajo, ejecutar la aplicación de entrenamiento, introducir por teclado uno a uno los parámetros de la aplicación, ubicar aquí y allá archivos para que la aplicación pudiese trabajar con ellos. En definitiva, todo este procedimiento se vuelve pesado para más de un entrenamiento, y es que se necesitan varios para poder contrastar entre ellos y elegir el mejor.

Con el objetivo de automatizar este proceso, se ha diseñado un modelo de directorio que contiene de forma organizada en carpetas, todos los archivos y aplicaciones necesarias para el entrenamiento. La característica principal de este modelo de organización es que permite gestionar la ejecución de las aplicaciones y todo lo relacionado con ellas mediante unos simples archivos script. Se podría decir que estos archivos hacen las veces de intérpretes entre el usuario y las aplicaciones de entrenamiento que este quiere ejecutar. Más en concreto, los scripts son archivos programados en *bash* que contienen una serie de comandos ejecutables en el terminal con los que se realizan todas las tareas asociadas a la ejecución de las aplicaciones. La principal tarea de estos archivos es pasar a las aplicaciones los valores de sus parámetros y ejecutarlas, aunque también se han diseñado para que realicen otras operaciones adicionales como medir el tiempo que tardan en ejecutarse o generar informes de resultados. Así pues, cada vez que se quiera ejecutar una aplicación de entrenamiento, se entrará en el archivo script correspondiente, se configurarán los parámetros de la aplicación y después se lanzará al terminal haciendo doble clic sobre él. Desde ese momento todo el proceso de introducir los parámetros a la aplicación, lanzar la aplicación y recopilar los resultados se hará de forma automática.

En la siguiente imagen, se muestra el esquema de carpetas que presenta el directorio de entrenamiento:

Figura 3.30: *Directorio de entrenamiento.*

A continuación se detallará el contenido de cada una de estas carpetas:

- **Scripts.**

  Esta es la carpeta más importante del directorio de entrenamiento y a partir de la cual se controlará todo. En su interior se encuentran los archivos script que permiten configurar y ejecutar cada una de las aplicaciones que intervienen en el entrenamiento. Existirá un archivo para cada aplicación.

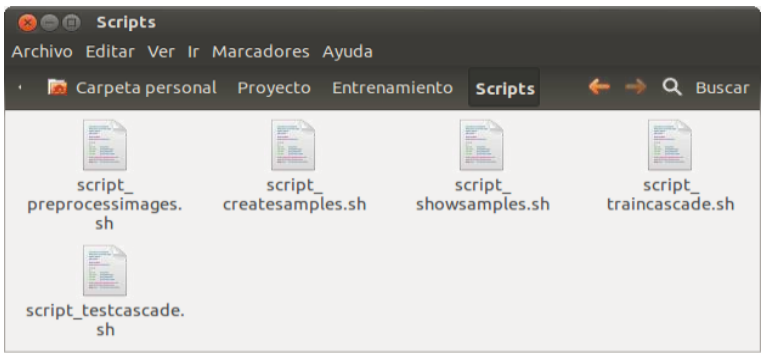


Figura 3.31: *Carpeta "Scripts".*

- **Aplicaciones.**

  Hace referencia a las aplicaciones *preprocessimages* y *testcascade* que se han desarrollado adicionalmente para el entrenamiento de clasificadores. En el interior de esta carpeta se tendrán ordenados los códigos fuente, los archivos de construcción, y los ejecutables de cada aplicación. No se tienen los ejecutables de las aplicaciones de *OpenCV* porque están instaladas en el directorio raíz junto con las librerías.

Figura 3.32: *Carpeta "Aplicaciones".*

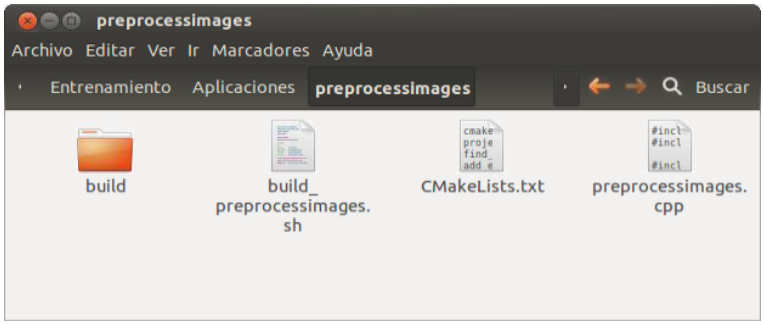


Figura 3.33: *Archivos de una aplicación.*

- **Imágenes.**

  Debe contener la colección de imágenes de muestra tanto para el entrenamiento como para la evaluación de los clasificadores. La colección de imágenes debe estar clasificada en dos carpetas según el tipo de muestras que sean: Positivas o Negativas. Servirá como ubicación para que las aplicaciones de entrenamiento puedan acceder a la colección de imágenes cuando lo necesiten.

Figura 3.34: *Carpeta "Imagenes".*

Figura 3.35: *Carpetas de una colección de imágenes.*

- **Listas.**

  En su interior se deben guardar los archivos de etiquetado también llamados aquí como listas de imágenes. Para cada aplicación se dispondrá de una carpeta donde se ubicarán los archivos de etiquetado requeridos por la misma.

Figura 3.36: *Carpeta "Listas".*

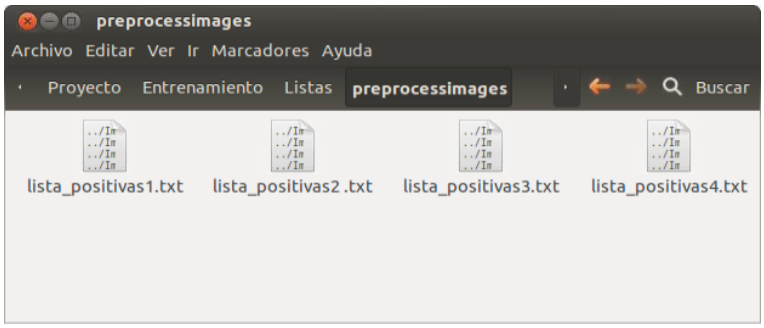


Figura 3.37: *Listas de una aplicación.*

- **Muestras.**

  Aquí se guardarán unos archivos con formato *.vec* que contienen las ventanas de las imágenes positivas donde se ubican los objetos a detectar. Estos archivos son generados por la aplicación *opencv_createsamples* para que sean utilizados después en el entrenamiento con *opencv_traincascade*.

Figura 3.38: *Carpeta "Muestras".*

- **Clasificadores.**

  En esta carpeta se irán almacenando los archivos en formato *.xml* que contienen a los clasificadores que han sido entrenados mediante la aplicación *opencv_traincascade*. También servirá para localizarlos cuando se les requiera en la aplicación *testcascade*.

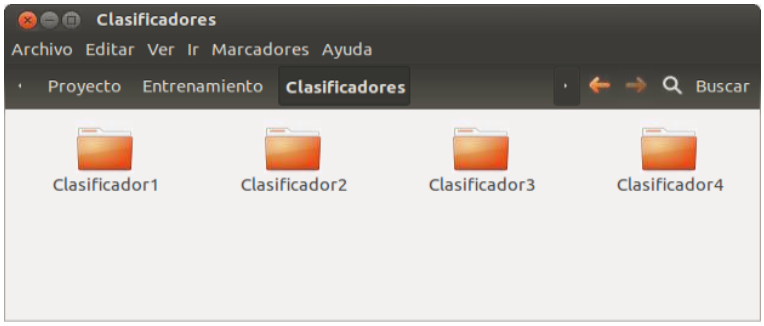


Figura 3.39: *Carpeta "Clasificadores".*

Figura 3.40: *Archivos de un clasificador.*

- **Curvas.**

  A esta carpeta van a parar los archivos de salida de la aplicación *testcascade*. Estos archivos contienen indicadores y datos para dibujar curvas con los que evaluar el rendimiento de los clasificadores ya entrenados.

Figura 3.41: *Carpeta "Curvas".*

- **Informes.**

  Esta carpeta sirve para almacenar unos archivos de texto que contienen informes con los resultados de la ejecución de cada una de las aplicaciones que intervienen en el entrenamiento. Los informes se organizan en carpetas teniéndose una para cada aplicación.

Figura 3.42: *Carpeta "Informes".*

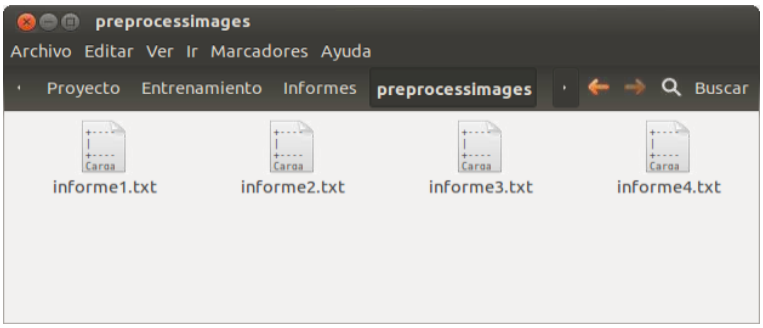


Figura 3.43: *Informes de una aplicación.*

3.2.5. Preprocesamiento de Imágenes

3.2.5.1. Descripción

El primer paso que se ha establecido para entrenar un clasificador es el de **preprocesar las imágenes** de muestra. La razón se debe a que en la mayoría de las ocasiones las imágenes recopiladas no están en las mejores condiciones porque puedan tener poco contraste, ruido, etc. Si en estas condiciones se entrenase al clasificador, seguramente que al algoritmo de aprendizaje le iba a costar más hacer que el clasificador reconociera al objeto buscado, además, la imagen que se crearía el clasificador del objeto sería algo menos definida, con unas características más amplias, por lo que a la hora de clasificar seguramente cometería más errores. Con el principal objetivo de reducir estas imperfecciones para que las imágenes tengan la mejor calidad posible y así sea más fácil y preciso entrenar al clasificador, se ha diseñado la aplicación *preprocessimages*.

La aplicación *preprocessimages* es un programa escrito en *C++* que utiliza las funciones que proporcionan las librerías de *OpenCV* para preprocesar las imágenes de muestra recopiladas. Las posibles operaciones que se pueden realizar sobre las imágenes con esta aplicación son las siguientes:

- **Redimensionar**. Esta operación cambia el tamaño de las imágenes en función del valor de un factor de redimensionado. Para ello hace uso de una interpolación lineal. Se integró con la idea de reducir el tamaño de aquellas imágenes que fuesen muy grandes para que el proceso de entrenamiento posterior se ejecutase más rápidamente.

- **Ecualizar**. Realiza una ecualización de las imágenes en base a su histograma. Se utiliza para mejorar aquellas imágenes con poco contraste.

- **Filtrar**. En esta operación se aplica un filtro bilateral sobre las imágenes para eliminar el ruido que presentan y manteniendo los bordes.

Hay que resaltar que para cualquiera de las operaciones elegidas, la aplicación convierte previamente las imágenes a escala de grises por lo que este será el formato de color de salida.

3.2.5.2. Parámetros

La configuración de los parámetros de la aplicación *prerpocessimages* se hará, como se vino explicando, desde un archivo *script* que en este caso se llama *script_preprocessimages.sh*. En este archivo se tiene a los parámetros que trabajarán sobre la aplicación representados como variables a las que se les debe asignar el valor del parámetro al que hacen referencia. Los parámetros asociados a la aplicación *preprocessimages* son los siguientes:

- **`application`**

  Con este parámetro se índica la ubicación y el ejecutable que contiene a la aplicación *preprocessimages*.

- **`report_file`**

  Se utiliza para indicar el directorio y el nombre del archivo de texto donde se guardarán los resultados mostrados por pantalla durante la ejecución del *script*.

- **`list_filename`**

  Este parámetro sirve para introducir la ubicación de un archivo de texto que contenga línea a línea las rutas y el nombre de archivo de las imágenes a preprocesar. Se podrán utilizar las listas de imágenes creados en el etiquetado porque siguen el mismo formato.

- **`preprocessed_images_directory`**

  Aquí se pide la ubicación de la carpeta donde se van a guardar las imágenes una vez preprocesadas.

- **`resize`**

  Sirve para indicar a la aplicación que debe realizar una operación de redimensionado sobre las imágenes de acuerdo a un factor.

- **`resize_factor`**

  Se utiliza para especificar el valor del factor de redimensionado que se quiere aplicar sobre las imágenes. Si no se específica este parámetro se utiliza un valor por defecto de 0.5, es decir, las imágenes se reducen a la mitad. Requiere de la activación del parámetro `resize`.

- **`equalize`**

  Indica a la aplicación que debe hacer una operación de ecualizado sobre las imágenes.

- **`filter`**

  Con este parámetro se está diciendo a la aplicación que debe realizar una operación de filtrado sobre las imágenes. Para ello, el tipo de filtro utilizado es el bilateral. Para entender cómo trabajar con este tipo de filtrado se puede ir a la documentación de *OpenCV* **[31]** y buscar la función *bilateralFilter()*.

- **`filter_diameter`**

  Es el diámetro que tienen las regiones de píxeles vecinos sobre las que se va aplicando el filtrado bilateral. Por defecto, la aplicación le asigna un valor de 5. Requiere que el parámetro `filter` sea verificado.

- **`filter_sigma_color`**

  Indica la desviación típica sobre la que se construye el filtro bilateral en el dominio del color. Por defecto tiene el valor utilizado es de 10. Necesita de la verificación del parámetro `filter`.

- **`filter_sigma_space`**

  Representa la desviación típica sobre la que se construye el filtro bilateral en el dominio del espacio. El valor usado por defecto es 10. Requiere que el parámetro `filter` sea verdadero.

- **`show`**

  Este parámetro indica a la aplicación que debe mostrar por pantalla el resultado de preprocesar cada una de las imágenes.

- **`show_factor`**

  Este es al factor con el que se redimensionan las imágenes para mostrarla. Si no se especifica, el factor utilizado es 1, lo que significa que las imágenes se muestran con su tamaño real. Necesita que se confirme al parámetro `show`.

- **`help`**

  Si se activa este parámetro, se estará indicando a la aplicación *preproccessimages* que muestre por pantalla información acerca del uso de sus parámetros.

3.2.5.3. Configuración

```
...
application="../Aplicaciones/preprocessimages/build/preprocessimages"
...
report_file="../Informes/preprocessimages/informe.txt"
...                                                                  #<default>
list_file_name="../Listas/preprocessimages/lista_positivas.txt"      #<null>
preprocessed_images_directory="../Imagenes/Coleccion"                #<null>
resize="true"                                                        #<true>
resize_factor=0.5                                                    #<0.5>
equalize="false"                                                     #<false>
filter="false"                                                       #<false>
filter_diameter=5                                                    #<5 >
filter_sigma_color=10                                                #<10>
filter_sigma_space=10                                                #<10>
show="false"                                                         #<false>
show_factor=1                                                        #<1>
help="false"                                                         #<false>
...
```

Figura 3.44: *Configuración de los parámetros de script_preprocessimages.*

Gracias a los archivos *script*, el proceso de configuración de los parámetros será tan sencillo como abrir el archivo *script_preprocessimages*, editar los valores de los parámetros conforme a los deseados y guardar los cambios. En la imagen de la ***Figura 3.44*** se muestra un ejemplo de configuración de este *script*.

3.2.5.4. Ejecución

Tras configurar el archivo *script*, ejecutarlo será algo tan sencillo como hacer doble clic sobre el archivo o lanzarlos desde la línea de comandos. A continuación se muestra un ejemplo de la información que se imprimiría en pantalla durante la ejecución del *script*:

```
+--------------------------------------------------------------------------+
|                          SCRIPT_PREPROCESSIMAGES                         |
+--------------------------------------------------------------------------+
Cargando parametros...Hecho

+--------------------------------------------------------------------------+
Creando el directorio de imagenes preprocesadas...Hecho

+--------------------------------------------------------------------------+
Ejecutando preprocessimages...

Parameters:

  List file name: ../Listas/preprocessimages/lista_positivas.txt
  Preprocessed images directory: ../Imagenes/Coleccion
  Resize: true
  Resize factor : 0.5
  Equalize: true
  Filter: true
  Filter diameter: 5
  Filter sigma color: 10
  Filter sigma space: 10
  Show: false
  Show factor: 1

Preprocess images...Done. Preprocessed 7972 images.

+--------------------------------------------------------------------------+
Tiempo de ejecución: 0h 1m 47s

+--------------------------------------------------------------------------+
Pulse enter para salir...
```

Figura 3.45: *Informe con los resultados de ejecutar script_preprocessimages.*

3.2.5.5. Resultados

Para el entrenamiento del clasificador de semáforos que nos ocupa, se han hecho varias operaciones de preprocesado sobre las imágenes de muestra recopiladas, con el fin de valorar el efecto que tiene este proceso sobre el rendimiento del clasificador. En los siguientes pasos del entrenamiento, se utilizarán las colecciones de imágenes preprocesadas para entrenar a varios clasificadores y así en la etapa de evaluación analizar que operaciones de preprocesado mejoraban la calidad del clasificador.

A continuación se explican las tres operaciones de preprocesado que se han aplicado sobre la colección de imágenes recopiladas:

- La primera de ellas consiste en aplicar un redimensionado a las imágenes con el objetivo de reducir su tamaño a la mitad.

- En otra operación de preprocesado, además de reducir las imágenes a la mitad de su tamaño, también se las ha sometido a un ecualizado.

- La última operación consiste en reducir las imágenes igual que antes, pero en este caso se les aplica después un filtro. Los parámetros del filtro utilizados son los de por defecto.

Al final de todo, se obtuvieron tres nuevas colecciones de imágenes preprocesadas que se utilizarán junto con la colección de imágenes originales para entrenar a cuatro clasificadores. Al analizar estos clasificadores se demostrará el efecto que tiene la etapa de preprocesado en el entrenamiento de los clasificadores.

3.2.6. Creación de Muestras

3.2.6.1. Descripción

Continuando con el esquema planteado para entrenar un clasificador, el siguiente paso en el proceso se correspondería con **crear las muestras** positivas para el entrenamiento. Esta tarea ya entraría dentro de las definidas por las aplicaciones de *OpenCV* para el entrenamiento de clasificadores. En concreto quién la ejecuta es la aplicación *opencv_createsamples*.

La función de la aplicación *opencv_createsamples* consiste en extraer de las imágenes positivas las ventanas que contienen a los objetos y guardarlas en un archivo con formato *.vec* que será después utilizado por la siguiente aplicación para entrenar al clasificador. Para ubicar las ventanas sobre las imágenes, la aplicación hace uso de los archivos lista de imágenes positivas, los cuales deben contener para cada imagen, la ruta de la imagen, el número de ventanas, la localización y el tamaño de dichas ventanas. En definitiva, obtiene unas muestras positivas donde únicamente aparece el objeto de interés.

3.2.6.2. Parámetros

En la documentación de *OpenCV* **[31]** se puede encontrar información acerca de los parámetros que posee la aplicación *opencv_createsamples* y el significado de cada uno de ellos. En este caso, no se hará uso de todos ellos por lo que tan solo se limitará a explicar los que realmente se han utilizado. En concreto, los parámetros con los que se trabaja sobre la aplicación *opencv_createsampels* a través del archivo *script_createsamples* son los siguientes:

- **`application`**

  En este parámetro se índica la ubicación y el nombre del ejecutable que contiene a la aplicación *opencv_createsamples*. Como la aplicación se construye en el directorio raíz, bastará con especificar el nombre de la misma para que sea reconocida.

- **report_file**

  Se utiliza para indicar el directorio y nombre del archivo en el que se guardará un informe con los resultados de la ejecución del *script*.

- **collection_file_name**

  Este parámetro sirve para especificar el archivo lista de imágenes positivas que se utiliza como referencia para extraer las ventanas que contienen a los objetos.

- **vec_file_name**

  Es el archivo en el que se guardan las muestras del objeto que se extraen de las imágenes positivas.

- **number_of_samples**

  Sirve para indicar el número total de muestras del objeto que se tienen en la colección de imágenes positivas. Por defecto se consideran 1000 muestras.

- **sample_width**

  Es el ancho con el que las muestras extraídas se redimensionan y almacenan en el archivo `vec_file_name`. El ancho predefinido es de 24 píxeles.

- **sample_height**

  En este caso, es el alto de las muestras extraídas con el que se redimensionan y almacenan en el archivo `vec_file_name`. Por defecto se utilizan 24 píxeles de alto.

Además de *script_createsamples*, se ha creado otro *script* llamado *script_showsamples* que permite mostrar las muestras creadas con el primero. Este *script* trabaja también sobre la aplicación *opencv_createsamples*, solo que en este caso el propio script se ha programado para cambiar el modo de funcionamiento de dicha aplicación. Los parámetros que se tienen para controlar esta nueva función son los siguientes:

- **application**

  Debe asignársele el nombre del archivo ejecutable que contiene a la aplicación *opencv_createsamples*.

- **vec_file_name**

  Se utiliza para especificar el archivo en el que se encuentran las muestras del objeto que se quieren visualizar.

- **scale_factor**

  Este parámetro sirve para indicar el factor de escala que se aplica a las muestras para visualizarlas.

- **sample_width**

  Es el ancho con el que las muestras han sido guardadas en el archivo `vec_file_name`. Por defecto se considera un ancho de 24 píxeles.

- **sample_height**

  Es el alto con el que las muestras han sido guardadas en el archivo `vec_file_name`. Por defecto se considera un alto de 24 píxeles.

3.2.6.3. Configuración

Para configurar los archivos *script* del directorio de entrenamiento, bastará con editar el contenido de los archivos *script_createsamples* y *script_showsamples*:

```
...
application="opencv_createsamples"
...
report_file="../Informes/createsamples/informe.txt"
...                                                                    #<default>
collection_file_name="../Listas/createsamples/lista_positivas.txt"     #<null>
vec_file_name="../Muestras/muestras.vec"                               #<null>
number_of_samples=1000                                                 #<1000>
sample_width=24                                                        #<24>
sample_height=24                                                       #<24>
...
```

Figura 3.46: *Configuración de los parámetros de script_createsamples.*

```
...
application="opencv_createsamples"
...                                                                    #<default>
vec_file_name="../Muestras/muestras.vec"                               #<null>
scale_factor=4                                                         #<4>
sample_width=24                                                        #<24>
sample_height=24                                                       #<24>
...
```

Figura 3.47: *Configuración de los parámetros de script_showsamples.*

3.2.6.4. Ejecución

A continuación, se muestra el contenido de un informe de ejecución de *script_createsamples*:

```
+--------------------------------------------------------------------------+
|                           SCRIPT_CREATESAMPLES                           |
+--------------------------------------------------------------------------+
Cargando parametros...Hecho

+--------------------------------------------------------------------------+
Ejecutando opencv_createsamples...

Info file name: ../Listas/createsamples/lista_positivas.txt
Img file name: (NULL)
Vec file name: ../Muestras/muestras.vec
BG  file name: (NULL)
Num: 7535
BG color: 0
BG threshold: 80
Invert: FALSE
```

```
Max intensity deviation: 40
Max x angle: 1.1
Max y angle: 1.1
Max z angle: 0.5
Show samples: FALSE
Width: 16
Height: 30
Create training samples from images collection...
Done. Created 7535 samples

+----------------------------------------------------------------------------+
Tiempo de ejecución: 0h 0m 20s

+----------------------------------------------------------------------------+
Pulse enter para salir...
```

Figura 3.48: *Informe con los resultados de ejecutar script_createsamples.*

3.2.6.5. Resultados

Hay que comentar que para el entrenamiento del clasificador de semáforos, la colección de imágenes positivas de que se dispone ha sido dividida en dos, una parte para realizar el entrenamiento y otra para testar el clasificador resultante. Así que de las 7972 imágenes positivas que se tienen, se han utilizado para el entrenamiento 3986, en las cuales hay un total de 7535 muestras de semáforos. En cuanto a las imágenes negativas de la colección se utilizarán todas para entrenar al clasificador. En este caso se tienen 4111 imágenes negativas.

Para comprobar la influencia que tiene sobre el clasificador el número de muestras positivas y el número de muestras negativas con las que se le entrena, se ha recopilado otra colección de imágenes que se añadirá a la que se tiene y en la que predominan las imágenes negativas frente a las positivas. En total, esta nueva colección tendrá 8376 imágenes positivas y 15210 imágenes negativas. De la misma manera, las imágenes positivas han sido divididas en dos, teniéndose para entrenar 4188 imágenes en las que aparecen 8036 muestras de semáforos.

Teniendo en cuenta todo esto, se han creado siete colecciones de muestras que cubren las diferentes opciones que hay:

- El primer grupo de muestras es el obtenido a partir de la nueva colección de muestras en la que predomina el número de imágenes negativas.

- Otros cuatro grupos de muestras surgen como resultado de obtener las muestras de las cuatro colecciones de imágenes que se tienen: originales, redimensionadas, ecualizadas y filtradas. Para todas ellas, se ha utilizado un tamaño de muestra de 16x30 porque es el que más se aproxima al valor por defecto manteniendo la relación de aspecto media de todas las muestras.

- Los dos siguientes grupos de muestras se generan a partir de la colección de imágenes originales y se obtienen para comprobar el efecto del tamaño de las muestras en el entrenamiento. Como antes ya se han obtenido muestras con un tamaño de 16x30, ahora se probará con un tamaño de muestra de la mitad, 8x15, y del doble, 32x60.

Así pues, hasta este punto se tienen siete colecciones de imágenes preparadas para entrenar.

3.2.7. Entrenamiento del Clasificador

3.2.7.1. Descripción

En esta parte del proceso de entrenamiento es en la que realmente se va a **entrenar al clasificador**. La aplicación encargada de realizar esta tarea es una de las proporcionadas por las librerías de *OpenCV* y se llama *opencv_traincascade*.

La aplicación *opencv_traincascade* está específicamente diseñada para el entrenamiento de clasificadores en cascada. Para ello, implementa varios tipos de algoritmos de entrenamiento, entre los que se puede elegir, que se basan en el método Boosting adaptado para clasificadores en cascada y en el que las etapas de la cascada se construyen como estructuras de clasificadores *weak* dispuestos en árbol. Los algoritmos entre los que se puede elegir son Discrete AdaBoost (DAB), Real AdaBoost (RAB), LogitBoost (LB) y Gentle AdaBoost (GAB). Estos algoritmos se valen de unas muestras positivas y negativas con las que se enseña al clasificador a distinguir el objeto de interés del resto de objetos. Para obtener los rasgos que caracterizan al objeto y con ellos enseñar al clasificador a reconocerlos, los algoritmos de entrenamiento utilizan dos técnicas de extracción de características que son las Características *Haar* y los Patrones Binarios Locales *(LBP)*.

3.2.7.2. Parámetros

Para poder realizar el entrenamiento de clasificadores desde el directorio de entrenamiento, será necesario describir los parámetros que se van a utilizar para controlar la ejecución de la aplicación *opencv_traincascade*. Estos parámetros serán configurados desde el archivo *script* asignado a esta aplicación y llamado como *script_traincascade*. Si se quisieran conocer los parámetros que trabajan directamente con la aplicación *opencv_traincascade*, se puede acudir a la documentación que proporciona *OpenCV* **[31]**. Los parámetros que contiene el archivo *script_traincascade* son los siguientes:

- **`application`**

  Es la ruta del archivo ejecutable donde se encuentra la aplicación opencv_traincascade. Como dicha aplicación está construida en el directorio raíz, basta con indicar solamente el nombre del archivo ejecutable.

- **`report_file`**

  Aquí se indica el archivo de texto sobre el que se escribirá un informe con los resultados de ejecutar el *script*. Estos resultados no son más que los datos que se imprimen por pantalla.

- **`cascade_dir_name`**

  Esta es la carpeta donde se almacenará el archivo en formato *.xml* que contiene al clasificador. También se irán guardando en ella las etapas que se vayan construyendo para poder reanudar el entrenamiento en caso de que se haya detenido. Si la carpeta no existe, el propio *script* se encarga de crearla.

- **vec_file_name**

  Es el archivo con formato *.vec* que se creó con la aplicación *opencv_createsamples* y que contiene el conjunto de muestras del objeto extraídas de la colección de imágenes positivas.

- **background_file_name**

  Este parámetro se utiliza para indicar el nombre del archivo en que el que se tienen listadas las imágenes negativas.

- **number_of_positive_samples**

  En este parámetro se ha de especificar el número de muestras del objeto de las que se dispone en el archivo `vec_file_name`. Por defecto se consideran 2000 muestras.

- **percentage_of_positive_samples_used_per_stage**

  Con este parámetro se debe especificar el porcentaje de muestras positivas que se utilizan para entrenar a cada una de las etapas del clasificador. Se aconseja utilizar un valor por debajo del 90 por ciento para que el entrenamiento no se vea interrumpido por falta de muestras.

- **number_of_negative_samples**

  Aquí se solicita el número de muestras o imágenes negativas de las que se dispone en el archivo `background_file_name`. Por defecto se consideran 1000 muestras.

- **number_of_stages**

  Sirve para indicar el número de etapas que debe tener el clasificador a construir. Puede suceder que el número de etapas del clasificador construido sea menor que el establecido debido a que el algoritmo de entrenamiento ha alcanzado los objetivos de clasificación antes. También puede darse el caso de que el número de etapas no sea suficiente para conseguir los objetivos de clasificación. El número de etapas por defecto es de 20.

- **precalculated_vals_buffer_size_in_Mb**

  En este parámetro se debe especificar el espacio de la memoria RAM del equipo en MB que se dedica para precalcular los valores de las características *Haar* o *LBP*. Cuanta mayor memoria se asigne, más rápido es el cálculo. Por defecto se utilizan 256 MB.

- **precalculated_idxs_buffer_size_in_Mb**

  En este parámetro se debe especificar el espacio de la memoria RAM del equipo en MB que se dedica para precalcular los índices de las características *Haar* o *LBP*. Cuanta mayor memoria se asigne, más rápido es el cálculo. Por defecto se utilizan 256 MB.

- **feature_type**

  Es el tipo de características que utilizará el clasificador para detectar los objetos. Se puede elegir entre características *Haar* o *LBP*. Por defecto se consideran características *Haar*.

- **sample_width**

  Se utiliza para indicar el ancho en pixeles de las muestras contenidas en el archivo `vec_file_name`. Por defecto se considera un ancho de 24 píxeles.

- **sample_height**

  Se utiliza para indicar el alto en pixeles de las muestras contenidas en el archivo `vec_file_name`. Por defecto se considera un alto de 24 píxeles.

- **boost_type**

  Con este parámetro se indica el tipo de algoritmo de entrenamiento utilizado. Se puede elegir entre los algoritmos Discrete AdaBoost (DAB), Real AdaBoost (RAB), LogitBost (LB) y Gentle AdaBoost (GAB). Por defecto se considera el algoritmo GAB.

- **min_hit_rate**

  Es la mínima tasa de éxito que el algoritmo de entrenamiento debe alcanzar en cada etapa. Mientras que en una etapa del clasificador no se consiga esta tasa, el algoritmo de entrenamiento continúa entrenándola. Por defecto se establece una tasa mínima de éxito de 0.995.

- **max_false_alarm_rate**

  Es la máxima tasa de falsas alarmas que el algoritmo de entrenamiento debe admitir en cada etapa. Mientras que en una etapa del clasificador no se consiga bajar de esta tasa, el algoritmo de entrenamiento continúa entrenándola. Por defecto se establece una tasa máxima de falsas alarmas de 0.5.

- **weight_trim_rate**

  Este parámetro viene a indicar las muestras que se ignoran en el proceso de entrenamiento para aumentar la velocidad del proceso. Si el valor del peso asignado a una muestra dividido por el máximo valor que puede adquirir es inferior a `weight_trim_rate` se considera que la muestra no es relevante y se excluye del entrenamiento. Se le asigna por defecto un valor de 0.9.

- **max_depth_of_weak_tree**

  Es la máxima profundidad en número de nodos que puede alcanzar un árbol de clasificadores *weak* durante el entrenamiento. El valor por defecto asignado es de 1, lo que significa que se utilizan clasificadores *stump*.

- **max_weak_tree_count**

  Representa el máximo número de árboles de clasificadores *weak* que puede contener cada etapa del clasificador. Por defecto se consideran 100 árboles.

- **haar_feature_mode**

  Sirve para seleccionar el set de características *Haar* a usar en el entrenamiento. Las opciones son BASIC en el que se usan solo características de dos regiones horizontales y verticales, CORE en la que se usan características de dos y tres regiones horizontales y verticales, y ALL que incluye todas las anteriores y además sus versiones rotadas a 45 grados. Por defecto la aplicación establece el set de características BASIC.

3.2.7.3. Configuración

Si se quieren configurar los parámetros de *script_traincascade*, no habrá más que acudir a la carpeta *Scripts* del directorio de entrenamiento y editar el archivo, teniendo en cuenta el significado de cada uno de los parámetros que aquí se ha descrito.

```
...
application="opencv_traincascade"
...
report_file="../Informes/traincascade/informe.txt"
...                                                          #<default>
cascade_dir_name="../Clasificadores/Clasificador"            #<null>
vec_file_name="../Muestras/muestras.vec"                     #<null>
background_file_name="../Listas/traincascade/lista_negativas.txt"   #<null>
number_of_positive_samples=2000                              #<2000>
percentage_of_positive_samples_used_per_stage=100            #<100>
number_of_negative_samples=1000                              #<1000>
number_of_stages=20                                          #<20>
precalculated_vals_buffer_size_in_Mb=256                     #256>
precalculated_idxs_buffer_size_in_Mb=256                     #<256>
feature_type="HAAR"                                          #<{ HAAR | LBP }>
sample_width=24                                              #<24 (default)>
sample_height=24                                             #<24 (default)>
boost_type="GAB"                                       #<{ DAB | RAB | LB | GAB (*) }>
min_hit_rate=0.995                                           #<0.995>
max_false_alarm_rate=0.5                                     #<0.5>
weight_trim_rate=0.95                                        #<0.95>
max_depth_of_weak_tree=1                                     #<1>
max_weak_tree_count=100                                      #<100>
haar_feature_mode="BASIC"                              #<{BASIC (*) | CORE | ALL}>
...
```

Figura 3.49: *Configuración de los parámetros de script_traincascade.*

3.2.7.4. Ejecución

Desde el momento en que se ejecute el archivo *script_traincascade* se podrá seguir el proceso de entrenamiento desde la ventana del terminal:

```
+--------------------------------------------------------------------------+
|                            SCRIPT_TRAINCASCADE                           |
+--------------------------------------------------------------------------+
Cargando parametros...Hecho

+--------------------------------------------------------------------------+
Creando el directorio del clasificador...Hecho

+--------------------------------------------------------------------------+
Calculando el numero de muestras positivas por etapa...Hecho
```

```
+--------------------------------------------------------------------------+
Ejecutando opencv_createsamples...

PARAMETERS:
cascadeDirName: ../Clasificadores/Clasificador
vecFileName: ../Muestras/muestras.vec
bgFileName: ../Listas/traincascade/lista_negativas.txt
numPos: 6781
numNeg: 4111
numStages: 20
precalcValBufSize[Mb] : 1024
precalcIdxBufSize[Mb] : 1024
stageType: BOOST
featureType: HAAR
sampleWidth: 16
sampleHeight: 30
boostType: GAB
minHitRate: 0.995
maxFalseAlarmRate: 0.5
weightTrimRate: 0.95
maxDepth: 1
maxWeakCount: 100
mode: BASIC

===== TRAINING 0-stage =====
<BEGIN
POS count : consumed   6781 : 6781
NEG count : acceptanceRatio    4111 : 1
Precalculation time: 0
+----+---------+---------+
|  N |    HR   |    FA   |
+----+---------+---------+
|   1|        1|        1|
+----+---------+---------+
|   2| 0.999115| 0.375821|
+----+---------+---------+
END>

===== TRAINING 1-stage =====
<BEGIN
POS count : consumed   6781 : 6787
NEG count : acceptanceRatio    4111 : 0.475205
Precalculation time: 1
+----+---------+---------+
|  N |    HR   |    FA   |
+----+---------+---------+
|   1|        1|        1|
+----+---------+---------+
|   2|        1|        1|
+----+---------+---------+
|   3| 0.997346| 0.472634|
+----+---------+---------+
END>

...

+--------------------------------------------------------------------------+
Tiempo de ejecución: 0h 2m 40s

+--------------------------------------------------------------------------+
Pulse enter para salir...
```

Figura 3.50: *Informe con los resultados de ejecutar script_traincascade.*

Como se puede comprobar en la imagen, la aplicación *opencv_traincascade* va imprimiendo por pantalla los resultados del entrenamiento de cada etapa. En la cabecera de las etapas aparece el número de imágenes tanto positivas como negativas que se van a emplear en entrenar la etapa y el tiempo que se ha tardado en precalcular los valores de las características. Después aparece una tabla en la que se representan el número de iteración que se está entrenando (N) y que coincide con el número clasificadores *weak* que hasta el momento forman la etapa del clasificador, y las tasa de éxito (Hit Rate) (HR) y falsa alarma (False Alarm) (FA) que se ha conseguido en la iteración. Se puede apreciar como el algoritmo de entrenamiento va añadiendo clasificadores *weak* entrenados a las etapas hasta que se consiguen las tasas fijadas.

3.2.7.5. Resultados

Con el objetivo de obtener el mejor clasificador de semáforos posible, los entrenamientos se han organizado en grupos en cada uno de cuales se evalúa un parámetro de los que intervienen en el proceso de entrenamiento. Aquellos parámetros que mejores resultados demuestren serán los utilizados para entrenar después al clasificador definitivo. A continuación se describen los grupos de clasificadores que han organizado y que parámetro se evalúa en cada uno de ellos.

- **Clasificador de referencia.** El primer clasificador que se ha entrenado sirve como referencia a la hora de comparar con el resto de clasificadores. Los parámetros con los que se ha entrenado son los siguientes: colección de imágenes originales, tamaño de muestras positivas de 16x30, características *Haar*, colección de características *Haar* BASIC, algoritmo de entrenamiento GAB, número de etapas 20, profundidad del árbol de clasificadores 1, tasa de éxito de 0.995 y tasa de falsa alarma de 0.5.

- **Número de muestras negativas.** En este grupo de clasificadores se intentará probar como influye en el rendimiento del clasificador el aumento del número de muestras negativas. Para ello se ha entrenado un clasificador con la colección de imágenes originales a la que se le ha añadido otra colección donde predominan las imágenes negativas. Este clasificador únicamente se comparará con el de referencia.

- **Preprocesado de las imágenes.** Aquí se intenta valorar el efecto que tiene preprocesar las imágenes antes de utilizarlas en el entrenamiento. Con este objetivo se han entrenado tres clasificadores: uno con la colección de imágenes reducidas a la mitad de tamaño, otro con la colección de imágenes ecualizadas y el último con la colección de imágenes filtradas. Para compararlos se utiliza el clasificador de referencia, el cual ha sido entrenado con imágenes no preprocesadas.

- **Tamaño de las muestras positivas.** En este grupo se han entrenado clasificadores para un tamaño de muestras de 8x15 y 32x60. Se completa el grupo con el clasificador de referencia entrenado con muestras de 16x30.

- **Tipo de características.** Lo forman clasificadores entrenados con los dos tipos de características que se tienen, *Haar* y *LBP*.

- **Colección de características *Haar*.** Para las características *Haar* se ha probado también entrenando a clasificadores con las distintas colecciones de características *Haar*: BASIC, CORE y ALL.

- **Algoritmo de entrenamiento.** En este grupo se han entrenado a los clasificadores mediante los distintos tipos de algoritmos que hay. Así se tiene clasificadores entrenados con Discrete AdaBoost (DAB), Real AdaBoost (RAB), LogitBoost (LB) y Gentle AdaBoost (GAB).

- **Número de etapas.** Aquí se tienen clasificadores entrenados con distinto número de etapas. Se han entrenado clasificadores con 18, 20 y 22 etapas.

- **Profundidad de los árboles.** Para valorar la importancia de la profundidad de los arboles de clasificadores *weak* se tienen clasificadores con 1, 2, 3 y hasta 4 niveles de profundidad.

- **Tasa de éxito.** Este grupo está formado por clasificadores entrenados con distintos valores de la tasa de éxito. Se han probado valores de 0.995, 0.999, 0.9995, 0.9999 y 0.99995.

- **Tasa de falsa alarma.** Lo componen clasificadores que han sido entrenados con valores de la tasa de falsa alarma de 0.5, 0.475, 0.45, 0.425 y 0.4.

Si se cuentan los clasificadores que hay en cada grupo, se tiene un total 26 clasificadores entrenados. Ahora faltaría que a estos clasificadores se les someta a un test para ver que parámetros son los más influyentes y que configuración es la mejor elección.

3.2.8. Evaluación del Clasificador

3.2.8.1. Descripción

Tras entrenar a un clasificador, la siguiente tarea que se debe hacer es la de **evaluar al clasificador**. Esto consiste en someter al clasificador a un estudio con el fin poder obtener una medida del rendimiento que presenta. Uno de los inconvenientes que tienen las librerías de *OpenCV* en el tema del entrenamiento de clasificadores, es que no incorporan ninguna herramienta que permita a los usuarios hacer este estudio. Por este motivo, se ha decidido desarrollar una aplicación propia que se encargue de realizar esta tarea y se llama *testcascade*.

La aplicación *testcascade* es un programa escrito en *C++* que se ayuda de las funciones de las librerías de *OpenCV* para poder trabajar con los clasificadores obtenidos del entrenamiento. En esta aplicación se tiene implementado un detector de objetos que hace uso de un clasificador entrenado para buscar los objetos de la colección de imágenes positivas que se le pasen. Los resultados obtenidos por el detector son comparados con los etiquetados sobre la misma colección de imágenes, obteniéndose así el rendimiento del clasificador.

Para entender de forma más detallada el funcionamiento de la aplicación *testcascade*, a continuación se describen paso a paso las operaciones que realiza:

- La aplicación comienza leyendo una a una las líneas de un archivo de etiquetado de las que extrae la ruta de la siguiente imagen a cargar y las marcas de los objetos que se han etiquetado en ella.

- Después se carga la imagen leída y se aplica sobre ella el detector de objetos que contiene al clasificador. Como resultado se obtienen las marcas de los objetos detectados.

- Aquellos marcas que estén próximas, se agrupan formando una única marca que será el promedio de ellas. A esta marca se le asigna una puntuación que es función de la cantidad de marcas que se han utilizado para formarla. Se entiende que si se obtienen muchas marcas juntas alrededor de una posición es muy probable que el objeto se encuentre en esa posición, de ahí la relación con la puntuación.

- Para eliminar aquellas marcas que se solapan entendiendo que corresponden al mismo objeto, se utiliza el método *non-maximun suppression*. Este método consiste en ir calculando el solapamiento que existe entre dos marcas, compararlo con un valor de umbral y si se supera eliminar aquella de menor puntuación. El solapamiento se calcula como el área de la intersección entre el área de la unión de las marcas. En este punto ya se tendrían las marcas detectadas definitivas.

- El siguiente paso consiste en comparar las marcas detectadas con las marcas etiquetadas. De mayor a menor puntuación, se va calculando el solapamiento entre las marcas detectadas y las marcas etiquetadas. Este solapamiento se calcula igual que antes, como la relación del área de intersección entre el área de unión de las marcas. Si este solapamiento supera un umbral y la marca etiquetada no se ha emparejado antes con otra marca detectada, se clasifica la marca detectada como verdadero positivo (*true positive*). En cualquier otro caso la marca detectada se clasifica como falso positivo (*false positive*). Al final de todo, si la marca etiquetada ha sido emparejada se clasifica como acierto (*hit*) y si no como fallo (*miss*).

- Después de haber clasificado tanto las marcas detectadas como las marcas etiquetadas, se hará un recuento del número de marcas que hay para cada clase.

- Una vez hechas estas operaciones para toda la colección de imágenes, se calculan las tasas de rendimiento, se obtienen los puntos de las curvas *ROC*, *DET* y *PR* y finalmente se procesan los indicadores de rendimiento. Los indicadores que se procesan para la curva *ROC* son el área bajo la curva *(AUC)* y la tasa de error igual *(EER)*, y para la curva *PR* son el área bajo la curva *(AUC)* y la precisión interpolada media *(API11)*.

- Finalmente tanto los indicadores de rendimiento como los puntos de las curvas son escritos en un archivo de texto.

Así pues, esta descripción ayudará a comprender el contenido de la aplicación *testcascade*.

3.2.8.2. Parámetros

Al igual que para todas las aplicaciones que intervienen en el entrenamiento de clasificadores, se tiene en el directorio de entrenamiento un archivo *script* llamado *script_testcascade* desde el que se gestionará todo lo relacionado con la aplicación *testcascade*. En el interior de este archivo se encuentran los parámetros con los que se trabajará sobre la aplicación y son los siguientes:

- **`application`**

  En este parámetro se ha de especificar la ruta del archivo ejecutable donde se implementa la aplicación *testcascade*.

- **`report_file`**

  Se utiliza para indicar la ruta del archivo en el que se guardará un informe con los resultados de la ejecución del *script* que se muestran en el terminal.

- **`cascade_file_name`**

  Es el nombre del archivo en formato *.xml* que contiene al clasificador que se quiere evaluar.

- **`list_file_name`**

  Aquí se indica el nombre del archivo lista que contiene las rutas de las imágenes positivas que se van a utilizar para evaluar al clasificador. Este archivo también debe incluir las marcas de los objetos que hay en las imágenes.

- **`data_file_name`**

  Este parámetro se utiliza para especificar la ubicación y nombre de un archivo en formato *.txt* donde se guardan los datos para dibujar las curvas de rendimiento obtenidas del test.

- **`scale_factor`**

  Como el detector busca objetos en las imágenes de entrada para distintas escalas, este parámetro indica cuanto se reducen estas imágenes en cada paso escala.

- **`min_object_width`**

  Es el ancho mínimo que debe tener el objeto detectado para considerarse como tal. Si el objeto detectado tiene menor ancho que este parámetro se ignora.

- **`max_object_width`**

  Es el ancho máximo que debe tener el objeto detectado para considerarse como tal. Si el objeto detectado tiene menor ancho que este parámetro se ignora.

- **`min_object_height`**

  Es el alto mínimo que debe tener el objeto detectado para considerarse como tal. Si el objeto detectado tiene menor ancho que este parámetro se ignora.

- **`max_object_height`**

  Es el alto máximo que debe tener el objeto detectado para considerarse como tal. Si el objeto detectado tiene menor ancho que este parámetro se ignora.

- **`group_threshold`**

  Este parámetro sirve para que el detector agrupe las marcas de los objetos detectados. Representa el mínimo número de vecinos que debe tener una marca para que no sea ignorada y se agrupe con sus vecinos. El mínimo número de vecinos por defecto es 1.

- **`min_nms_overlap`**

  Es el mínimo porcentaje de solapamiento existente entre dos marcas detectadas para que se consideren coincidentes. Si dos marcas son coincidentes se elimina la que tenga menor puntuación. El solapamiento entre marcas se mide como la relación del área de la intersección entre el área de la unión. Por defecto se considera 0.5

- **`min_match_overlap`**

  Indica el mínimo porcentaje de solapamiento que debe existir entre la marca de un objeto detectado y la marca de un objeto etiquetado para que se consideren coincidentes. Aquí el solapamiento entre marcas también se mide como la relación del área de la intersección entre el área de la unión. El valor por defecto es de 0.5.

- **`curve_type`**

  Con este parámetro se especifica el tipo de curvas de rendimiento que se quieren calcular y cuyos datos se guardarán en el archivo `data_file_name`. Las opciones son curva *ROC* (roc), curva *DET* (det), curva *PR* (pr) o todas (all). Por defecto el valor asignado es all.

- **`show`**

  Sirve para indicar a la aplicación *testcascade* que muestre por pantalla el proceso de evaluación del clasificador.

- **`help`**

  Si se activa este parámetro, se imprimirá por pantalla información de ayuda acerca de cómo usar los parámetros de la aplicación *testcascade*.

3.2.8.3. Configuración

Ahora que se sabe el significado de los parámetros de *script_testcascade*, configurarlos. Esto se hace abriendo el archivo *script*, dando a los parámetros lo valores deseados.

```
...
application="../Aplicaciones/testcascade/build/testcascade"
...
report_file="../Informes/testcascade/informe.txt"
...                                                          #<default>
cascade_file_name="../Clasificadores/Clasificador/cascade.xml"  #<null>
list_file_name="../Listas/testcascade/lista_positivas.txt"   #<null>
data_file_name="../Curvas/curvas.txt"                        #<null>
scale_factor=1.1                                             #<1.1>
min_object_width=0                                           #<0>
min_object_height=0                                          #<0>
max_object_width=0                                           #<0>
max_object_height=0                                          #<0>
group_threshold=1                                            #<1>
min_nms_overlap=0.5                                          #<0.5>
min_match_overlap=0.5                                        #<0.5 >
curve_type="all"                                  #<{ roc | det | pr | all (*) | null }>
show="false"                                                 #<false>
help="false"                                                 #<false>
...
```

Figura 3.51: *Configuración de parámetros de script_testcascade.*

3.2.8.4. Ejecución

Durante el proceso de evaluación del clasificador se irán imprimiendo por pantalla los resultados que se van obteniendo (ver ***Figura 3.52***). Lo primero que se muestra son los parámetros de la aplicación y los valores que se les ha pasado. Después comienza el test y se va imprimiendo en pantalla una tabla en la que aparecen por columnas: el nombre del archivo imagen sobre la que se está evaluando el clasificador (File Name) y el número de verdaderos positivos (TP), falsos negativos (FN) y falsos positivos (FP) que se han clasificado en dicha imagen. Una vez evaluada toda la colección de imágenes, al final se hace un recuento del número de objetos totales que hay en cada clase.

```
+--------------------------------------------------------------------------+
|                            SCRIPT_TESTCASCADE                            |
+--------------------------------------------------------------------------+
Cargando parametros...Hecho

+--------------------------------------------------------------------------+
Ejecutando testcascade...

Parameters:

  Cascade file name: ../Clasificadores/Clasificador/cascade.xml
  List file name: ../Listas/testcascade/lista_positivas.txt
  Data file Name: ../Datos/datos.txt
  Scale factor: 1.1
  Min objects size: 0x0
  Max objects size: 0x0
  Group threshold: 1
  Min nms overlap: 0.5
  Min match overlap: 0.5
  Curve type: all
  Show test: false
```

```
Cascade test...

  +----------------------+--------+--------+--------+
  | File Name            |     TP |     FN |     FP |
  +----------------------+--------+--------+--------+
  | img_10365.jpg        |      2 |      0 |      1 |
  +----------------------+--------+--------+--------+
  | img_10366.jpg        |      2 |      0 |      1 |
  +----------------------+--------+--------+--------+
  | img_10367.jpg        |      2 |      0 |      1 |
  +----------------------+--------+--------+--------+
  | img_10368.jpg        |      2 |      0 |      1 |
  +----------------------+--------+--------+--------+
  ...
  +----------------------+--------+--------+--------+
  | img_20765.jpg        |      3 |      0 |      1 |
  +----------------------+--------+--------+--------+
  | img_20766.jpg        |      3 |      0 |      1 |
  +----------------------+--------+--------+--------+
  | img_20767.jpg        |      3 |      0 |      3 |
  +----------------------+--------+--------+--------+
  | img_20768.jpg        |      3 |      0 |      0 |
  +----------------------+--------+--------+--------+
  | Total                |   5939 |    462 |   7321 |
  +----------------------+--------+--------+--------+

Curves calculation...Done

+--------------------------------------------------------------------------------+
Tiempo de ejecución: 0h 9m 16s

+--------------------------------------------------------------------------------+
Pulse enter para salir...
```

Figura 3.52: *Informe con los resultados de ejecutar script_testcascade.*

3.2.8.5. Resultados

Tras acabar la evaluación de un clasificador, en la carpeta *Curvas* del directorio de entrenamiento aparece un archivo de texto que contiene los indicadores y puntos de las curvas de rendimiento calculados durante el proceso*.*

```
ROC                             DET                             PR
AUC             EER                                             AUC             AP11
0.857141        0.179210                                        0.848723        0.828417
FPR             TPR             FPR             TNR             REC             PRE
0.000000        0.000000        0.000000        1.000000        0.000000        1.000000
0.000000        0.000156        0.000000        0.999844        0.000156        1.000000
0.000000        0.000312        0.000000        0.999688        0.000312        1.000000
0.000000        0.000469        0.000000        0.999531        0.000469        1.000000
0.000000        0.000625        0.000000        0.999375        0.000625        1.000000
...
0.999454        0.927824        0.999454        0.072176        0.927824        0.448024
0.999590        0.927824        0.999590        0.072176        0.927824        0.447990
0.999727        0.927824        0.999727        0.072176        0.927824        0.447956
0.999863        0.927824        0.999863        0.072176        0.927824        0.447922
1.000000        0.927824        1.000000        0.072176        0.927824        0.447888
```

Figura 3.53: *Archivo con los datos de las curvas.*

El archivo se organiza de tal manera que para cada tipo de curva se reservan dos columnas datos en las que la primera línea se indica el tipo de curva, en la segunda y tercera línea aparecen el tipo de indicadores de rendimiento calculados para esa curva y sus valores correspondientes, en la cuarta línea se tiene los nombres de las variables que representan cada una de las coordenadas de los puntos y en la quinta y sucesivas los puntos de las curvas. Darse cuenta que estos datos se organizan en columnas para que sea fácil transportarlos a una hoja de cálculo y desde allí dibujar las curvas.

Del proceso de entrenamiento para el clasificador de semáforos, se obtuvieron en total 26 clasificadores que, si se recuerda, estaban organizados en grupos de entrenamiento, de tal manera que en cada grupo se pretendía probar el efecto que tiene sobre el rendimiento del clasificador uno de los parámetros que intervienen en el entrenamiento. La medida de este efecto se obtiene de evaluar cada uno de los clasificadores y de compararlos con los otros del mismo grupo. Así pues, los clasificadores entrenados se han sometido a un test mediante la aplicación *testcascade* de la cual se han obtenido los archivos con los datos de las curvas de rendimiento. Después, el contenido de estos archivos se ha llevado a las hojas de cálculo de *Microsoft Office Excel* y, por grupos de entrenamiento, se han graficado las curvas junto con sus correspondientes indicadores. Para todos los clasificadores se han obtenido las tres posibles curvas que son la *ROC*, la *DET* y la *PR*. Dado que la duración de los entrenamientos también es un factor a tener en cuenta en la evaluación del rendimiento, de los informes se han extraído los tiempos de entrenamiento y se han graficado también por grupos. En definitiva, a continuación se tienen todos los gráficos de rendimiento obtenidos:

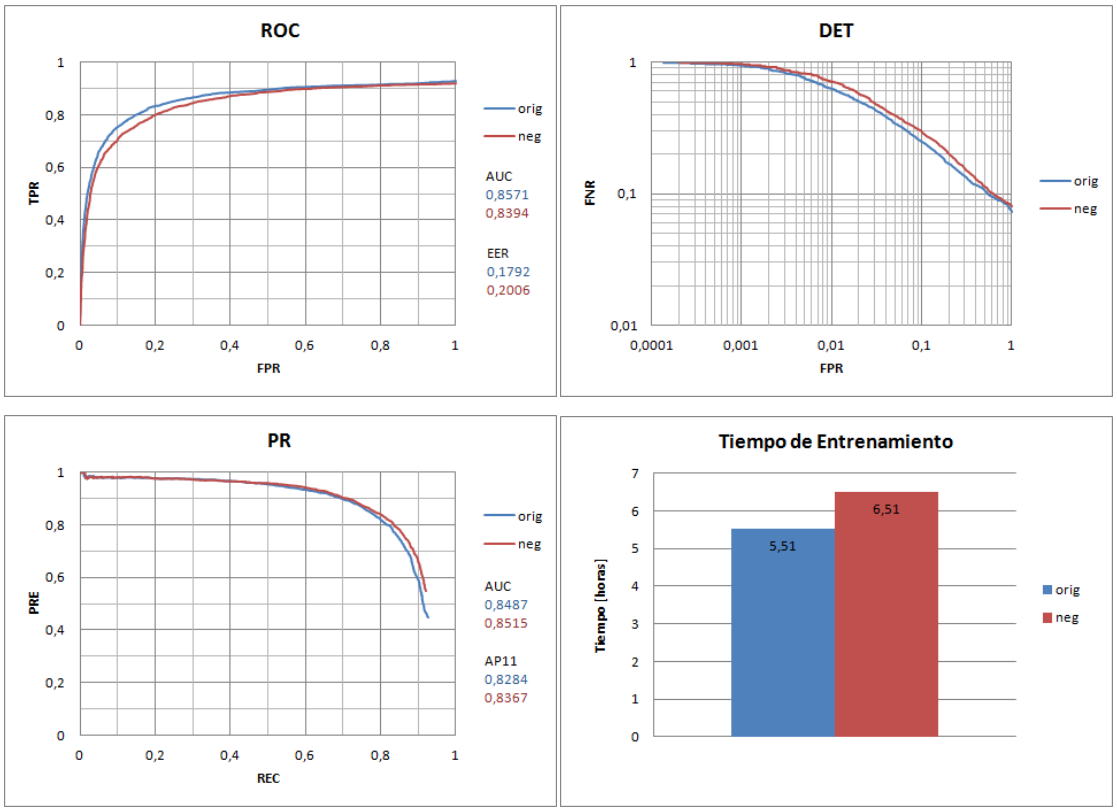


Figura 3.54 *Gráficos de rendimiento para evaluar el efecto que tiene el "número de imágenes negativas" sobre el rendimiento del clasificador.*

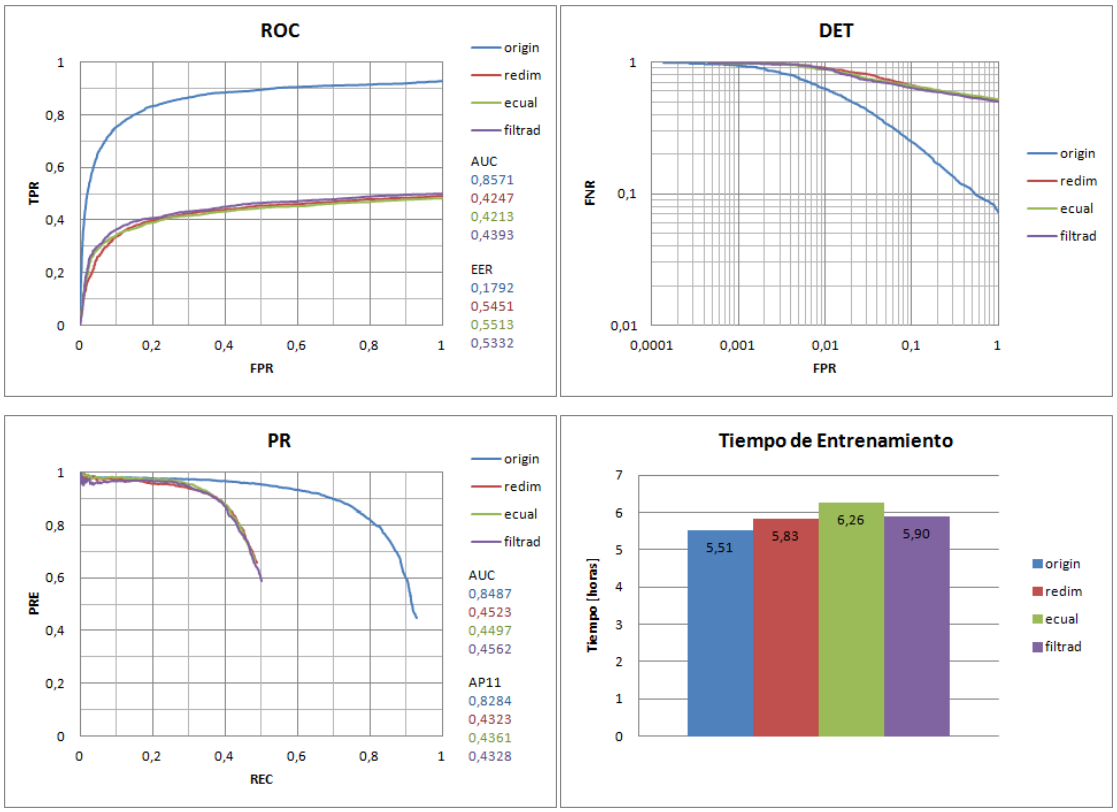


Figura 3.55: *Gráficos de rendimiento para evaluar el efecto que tiene el "preprocesado de las imágenes" sobre el rendimiento del clasificador.*

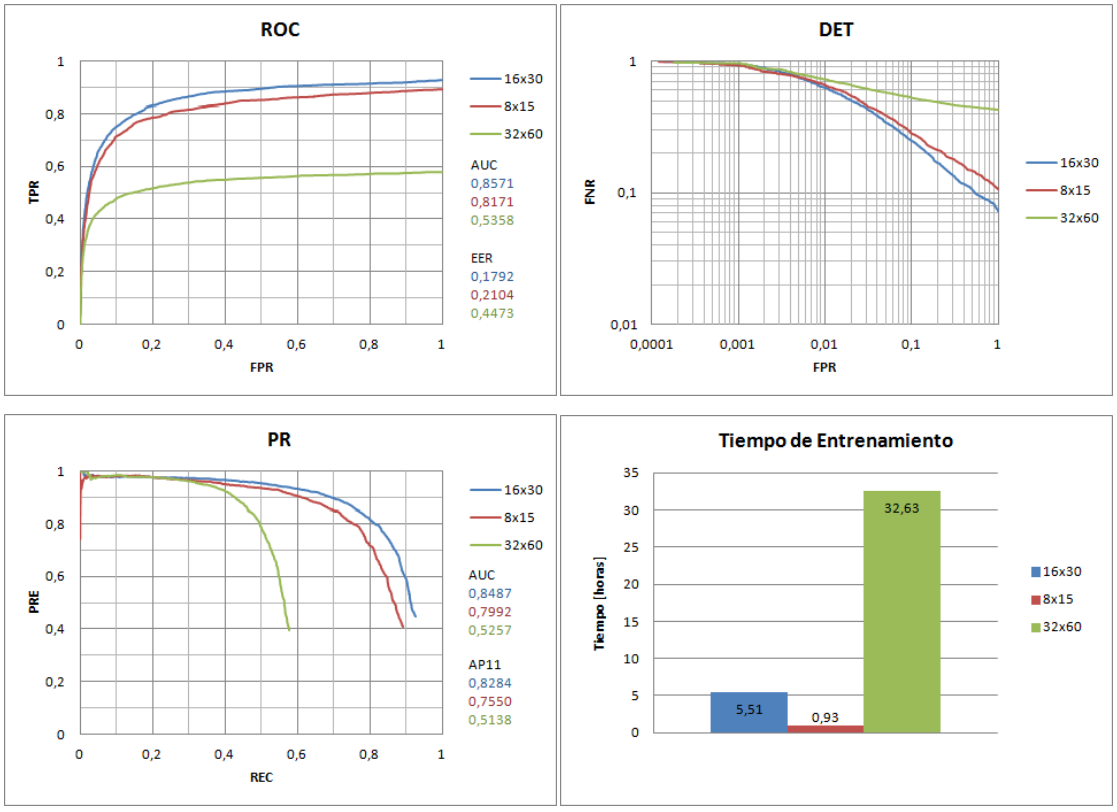


Figura 3.56: *Gráficos de rendimiento para evaluar el efecto que tiene el "tamaño de las muestras positivas" sobre el rendimiento del clasificador.*

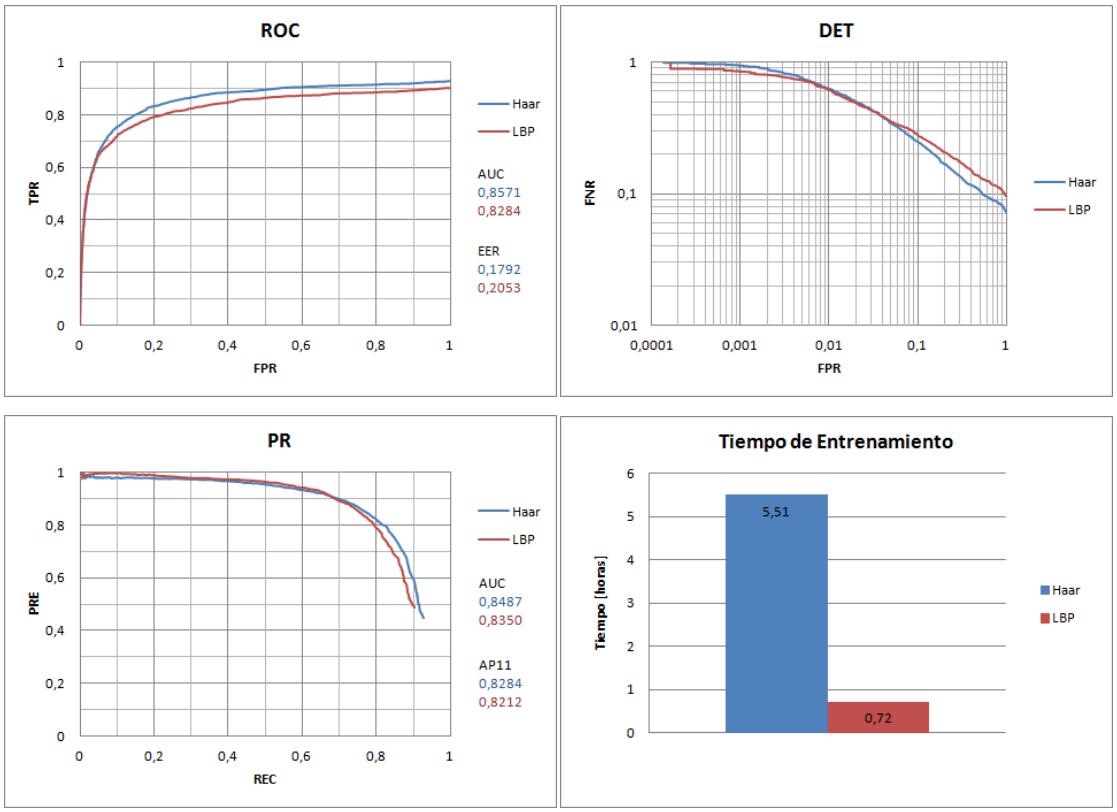


Figura 3.57: *Gráficos de rendimiento para evaluar el efecto que tiene el "tipo de características" sobre el rendimiento del clasificador.*

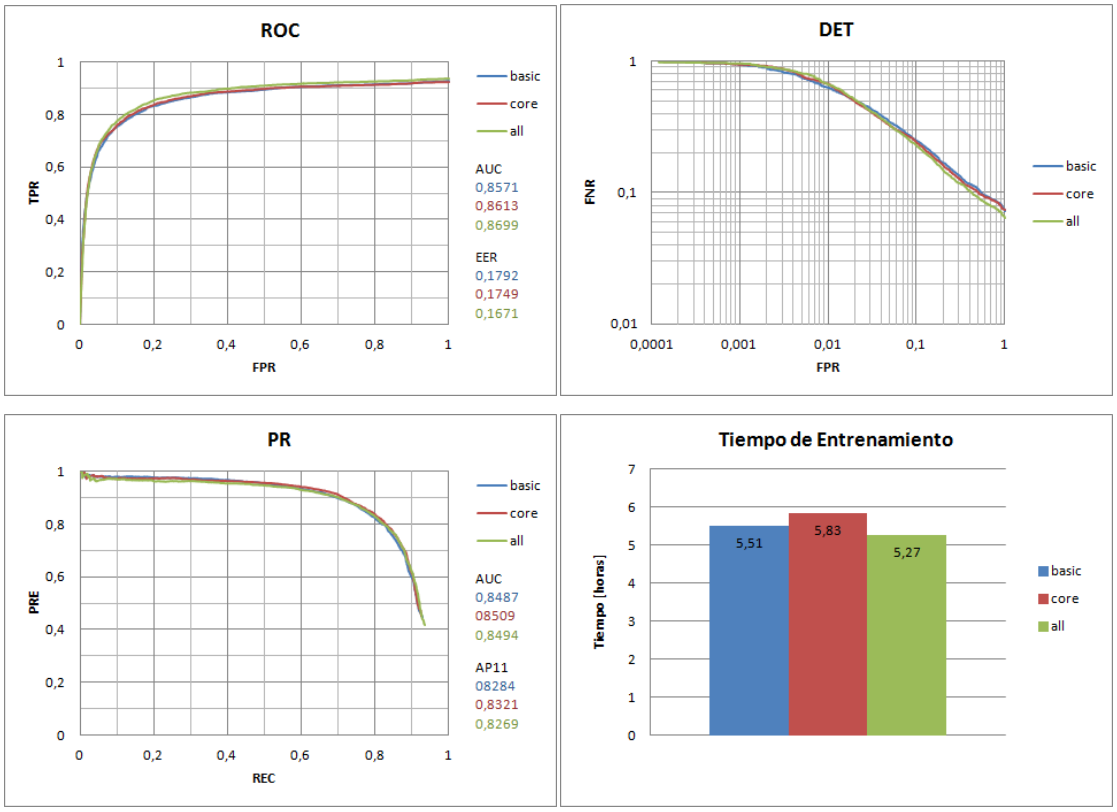


Figura 3.58: *Gráficos de rendimiento para evaluar el efecto que tiene la "colección de características Haar" sobre el rendimiento del clasificador.*

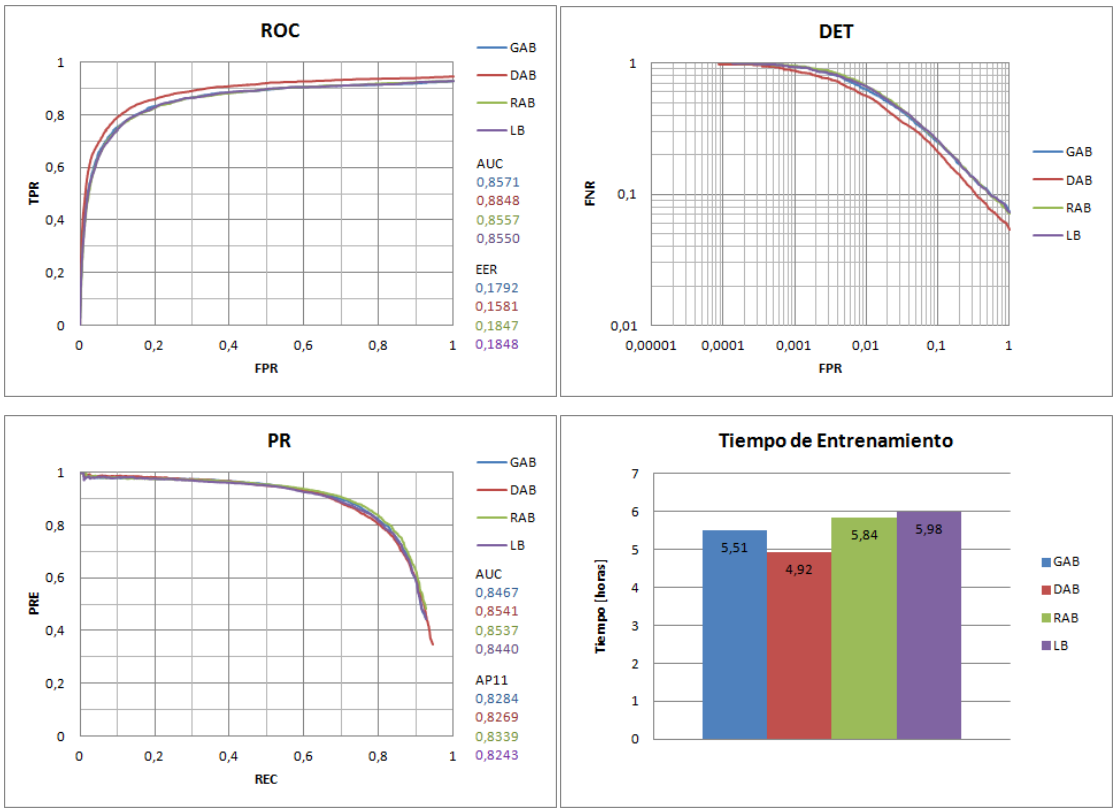


Figura 3.59: *Gráficos de rendimiento para evaluar el efecto que tiene el "algoritmo de entrenamiento" sobre el rendimiento del clasificador.*

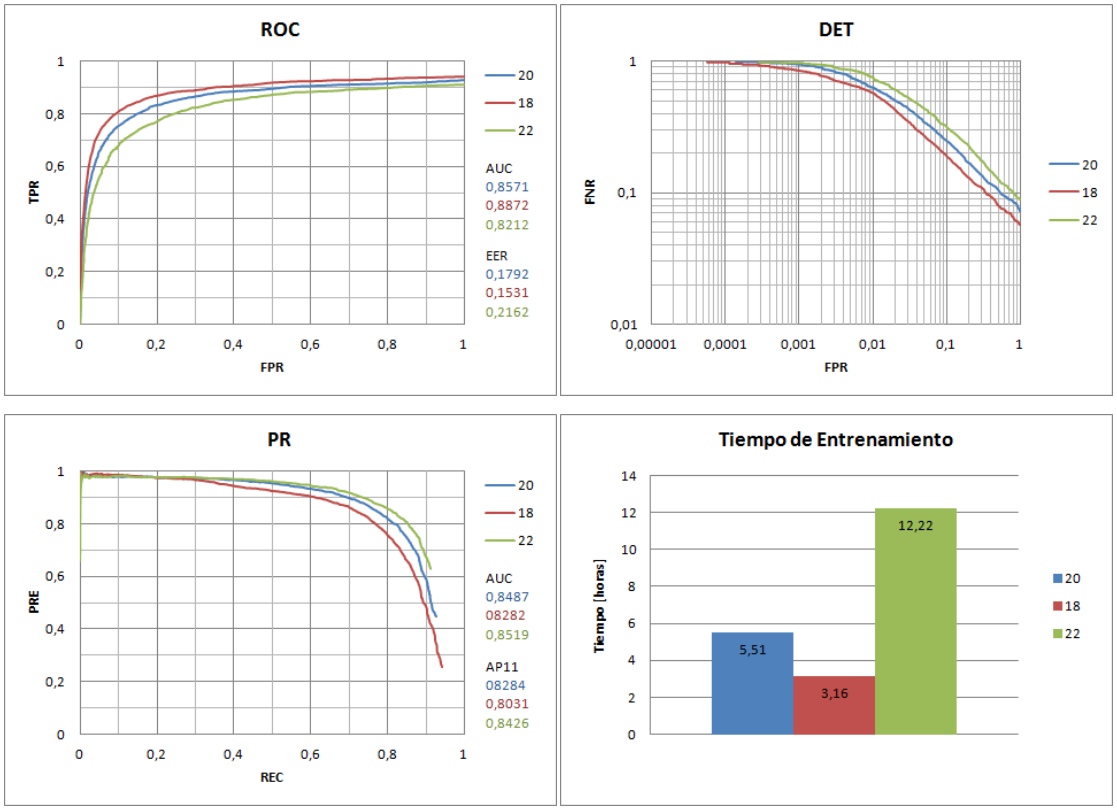


Figura 3.60: *Gráficos de rendimiento para evaluar el efecto que tiene el "número de etapas" sobre el rendimiento del clasificador.*

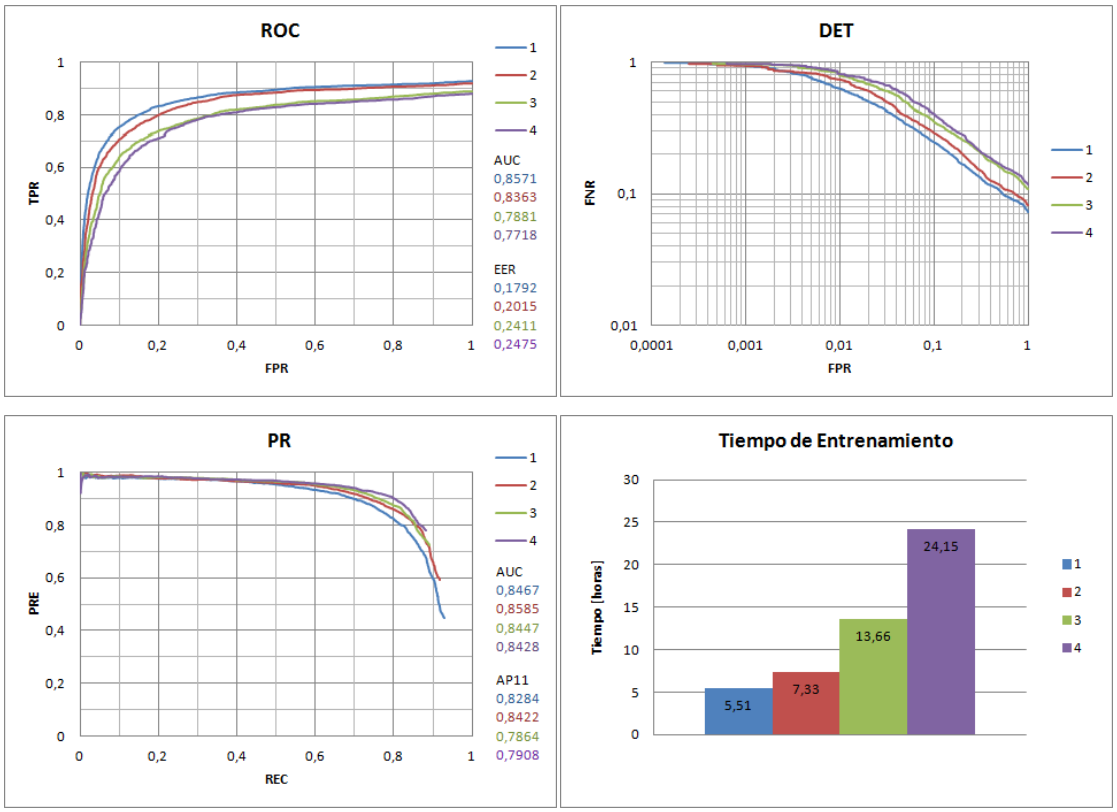


Figura 3.61: *Gráficos de rendimiento para evaluar el efecto que tiene la "profundidad de los árboles" sobre el rendimiento del clasificador.*

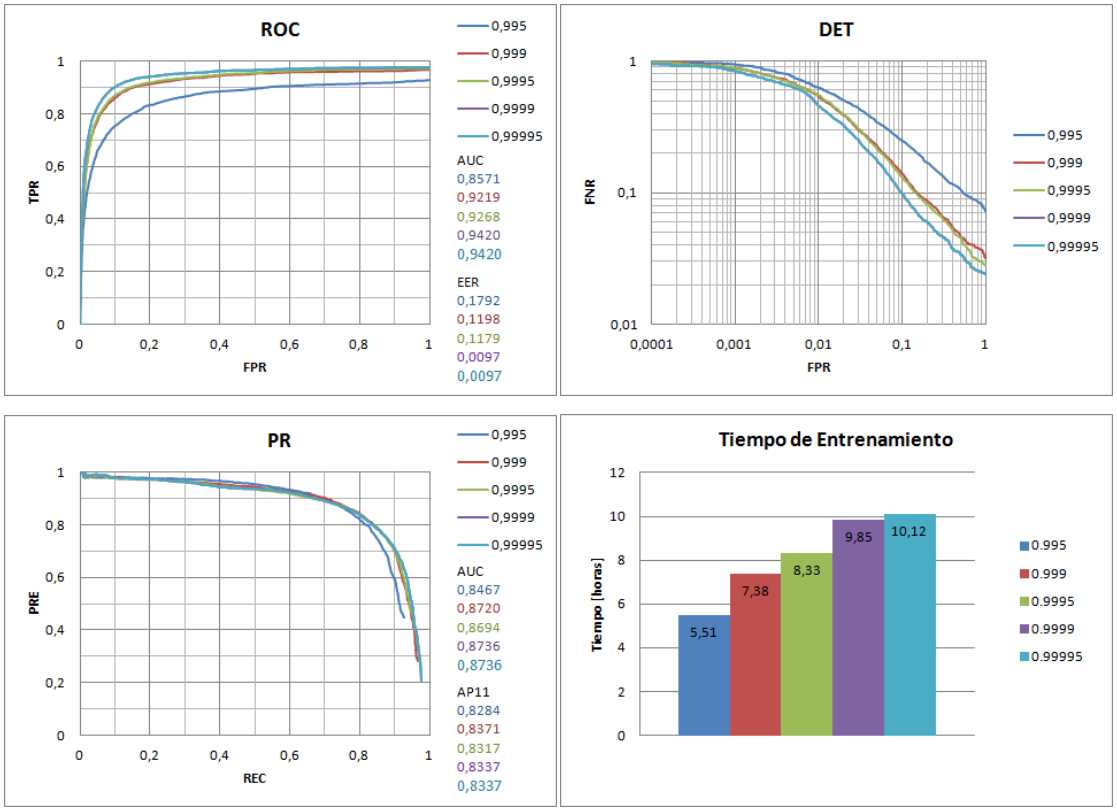


Figura 3.62: *Gráficos de rendimiento para evaluar el efecto que tiene la "tasa de éxito" sobre el rendimiento del clasificador.*

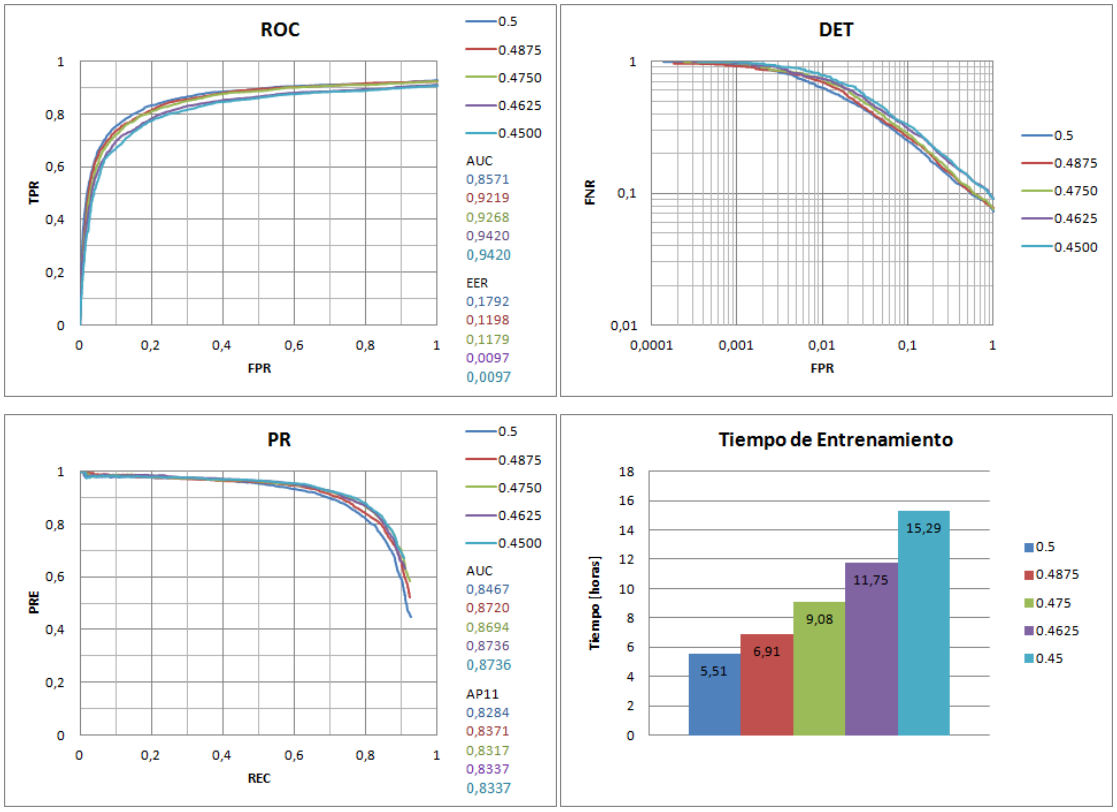


Figura 3.63: *Gráficos de rendimiento para evaluar el efecto que tiene la "tasa de falsa alarma" sobre el rendimiento del clasificador.*

3.2.9. Resultados del Entrenamiento

Gracias a los gráficos de rendimiento obtenidos en el proceso de evaluación, se ha podido estudiar cuál es el comportamiento de los clasificadores y como su rendimiento se ve influenciado por los parámetros del entrenamiento con los que han sido construidos. De este estudio se ha podido conocer que parámetros son los más significativos y que valores de éstos son los que mejores resultados proporcionan.

Para el detector de semáforos, se han decidido crear dos tipos de clasificadores definitivos, uno que utiliza características *Haar* y otro que hace lo propio con características *LBP*. La razón es que aunque el clasificador con características *Haar* presenta mejor rendimiento, el clasificador con características *LBP* tiene la ventaja de ser siempre más rápido. Si en un futuro se necesitase un clasificador rápido aún con no tan buen rendimiento, el clasificador *LBP* sería la solución.

Del análisis de los gráficos de rendimiento, los valores de los parámetros de entrenamiento que mejores resultados han dado y que por esa razón finalmente se han elegido son los siguientes:

- **Colección de imágenes:** originales.
- **Imágenes preprocesadas:** si, reducidas y filtradas.
- **Tamaño de las muestras positivas:** 16x30.

- **Tipo de caracterísiticas:** Haar y LBP.
- **Colección de características Haar:** all.
- **Algoritmo de entrenamiento:** DAB.
- **Número de etapas:** 20.
- **Profundidad de los árboles:** 2.
- **Tasa de éxito:** 0.9995.
- **Tasa de falsa alarma:** 0.475.

Una vez entrenados los clasificadores con los parámetros indicados, se han sometido al test obteniéndose los siguientes gráficos de rendimiento:

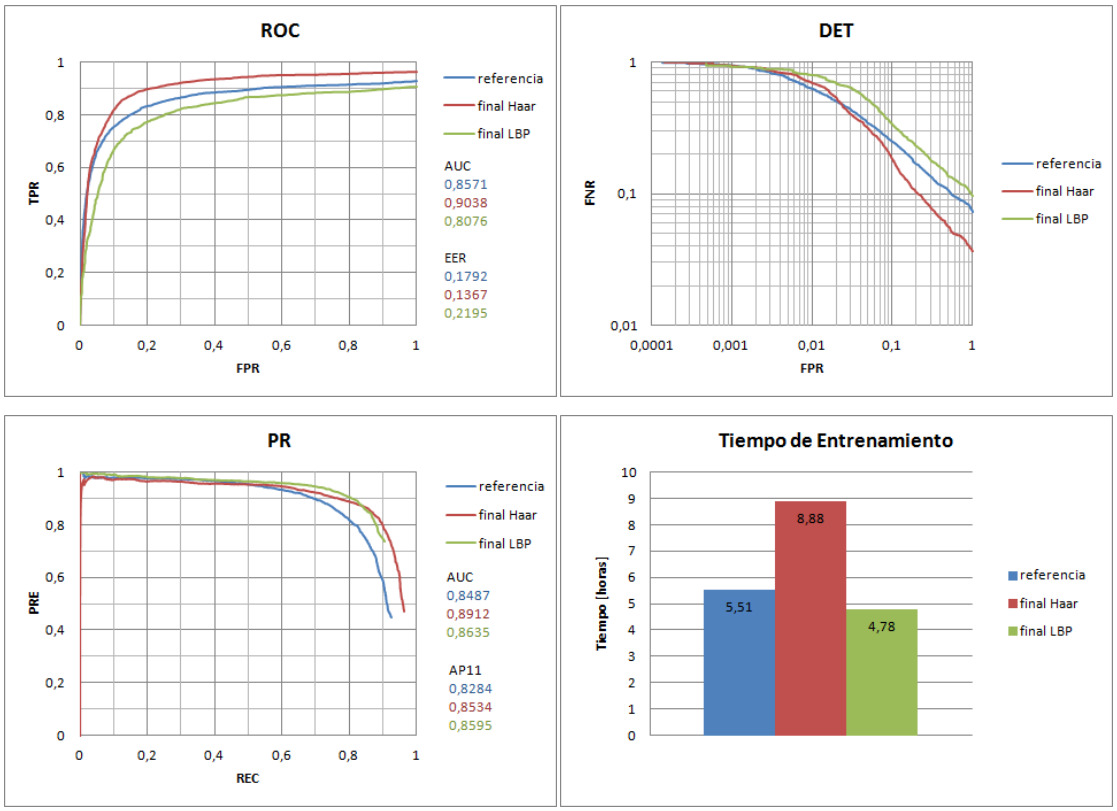


Figura 3.64: *Gráficos de rendimiento de los clasificadores finales.*

Como se pude comprobar en los gráficos y en los indicadores, el clasificador entrenado con características *Haar* es el que mejores resultados da para las tres curvas de rendimiento. Si se comparan estos resultados con los del clasificador de referencia se llega a la conclusión de que la elección de los valores de los parámetros de entrenamiento ha sido acertada. El único pero se encuentra en el tiempo de entrenamiento que es casi del doble que el del resto, lo que viene a indicar que es más lento de ejecutar. Por el contrario, se tiene al clasificador *LBP* que en general tiene peores resultados pero como se muestra en el gráfico de tiempo, para los mismos parámetros de entrenamiento que para el clasificador Haar tarda la mitad de tiempo. Esto claramente es un indicativo de lo rápido que es. En definitiva, si se quiere precisión se elegirá el clasificador Haar y si se quiere velocidad se elegirá el clasificador *LBP*.

3.3. Diseño de la Aplicación para la Detección y Reconocimiento de Semáforos

3.3.1. Introducción

Hasta este punto, el desarrollo del proyecto ha consistido en recopilar imágenes de muestra de los semáforos y entrenar con ellas a un clasificador para que tenga el mejor rendimiento posible. Como ya se vino comentando, este clasificador será utilizado por un sistema detector que indicará sobre las imágenes que se le pasen donde se ubican los semáforos. Ahora lo que se necesita es implementar dicho detector de semáforos. Pero diseñar un sistema detector de semáforos no sería suficiente para cumplir con los objetivos del proyecto. Este sistema ubicaría al semáforo dentro de las imágenes pero no sería capaz de interpretar las señalizaciones que emite. Para ello sería necesario integrar junto con el detector de semáforos otro sistema que permitiese, una vez detectado el semáforo, reconocer esas señales.

Así pues en este último apartado del desarrollo del proyecto, se presenta la aplicación que ha sido diseñada para la **detección y reconocimiento de semáforos** y que, como su nombre indica, permite tanto localizar los semáforos sobre las imágenes de entrada que se le pasen como interpretar las señales que estos emiten. La aplicación ha sido programada en *C++* haciendo uso de las funciones que proporcionan las librerías de *OpenCV*. Para entender su funcionamiento, en este apartado se pretende hacer una descripción paso a paso de cada una de las partes de las que se compone la aplicación, que funciones se utilizan en cada una ellas y con qué parámetros trabaja.

3.3.2. Descripción General

El código fuente de la aplicación que se ha creado para la detección y reconocimiento de semáforos se encuentra bajo el nombre de *tldr.cpp* (*traffic lights detection and recognition*). Como la extensión del archivo indica, la aplicación ha sido programada en *C++* y se ayuda de las funciones implementadas en las librerías de *OpenCV* para el procesamiento de imágenes. Haciendo una descripción general, el funcionamiento de la aplicación comienza captando, una a una, imágenes desde una fuente de entrada que bien puede ser o un archivo de texto con una lista de imágenes, o un archivo de vídeo o una cámara. Sobre cada imagen captada, se va aplicando un detector, que incorpora el clasificador de semáforos entrenado, con el que se ubican las ventanas de la imagen en las que se determina que hay semáforos.

Figura 3.65: *Detección de semáforos.*

Con los semáforos localizados, lo siguiente que hace es buscar donde se encuentra la luz dentro de la ventana que enmarca al semáforo. Para ello, utiliza un conjunto de técnicas de análisis morfológico que buscan la forma circular de la luz.

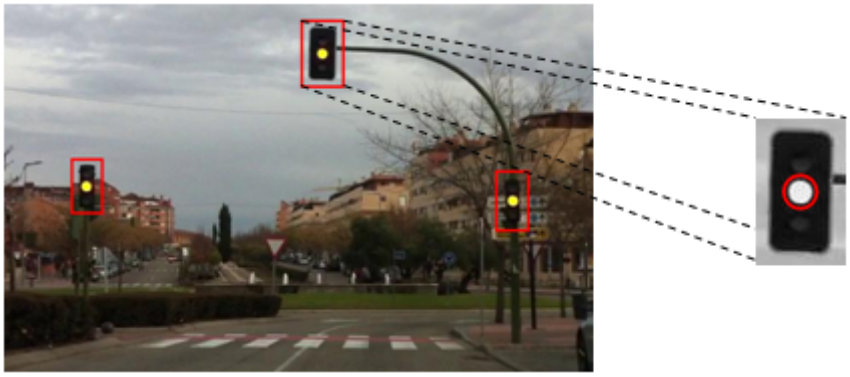


Figura 3.66: *Detección de la luz.*

Una vez localizados tanto el semáforo como la luz de este, la aplicación realiza un reconocimiento del color de la luz. Este procedimiento lo puede realizar de tres formas diferentes según elija el usuario. Las dos primeras que se explican se hacen con la imagen de entrada en escala de grises mientras que la tercera utiliza el espacio de colores HSL.

- La primera opción es la de reconocer el color de la luz (rojo, ámbar o verde) en función de la posición de ésta (arriba, centro o abajo) respecto del semáforo.

Figura 3.67: *Reconocimiento del color de la luz por posición.*

- Otra opción es aplicar la técnica de *matching templates* en la que se tienen una serie de plantillas con las distintas posiciones de la luz respecto del semáforo (de nuevo arriba, centro o verde), de tal forma que aquella plantilla que más coincida sobre el semáforo será la que indique la posición de la luz y por tanto el color (respectivamente rojo, ámbar o verde).

Figura 3.68: *Reconocimiento del color de la luz por matching.*

- La última posibilidad es identificar el color de la luz directamente sobre el canal de tono (*hue*) de la imagen, del que se extraerá el color medio de los pixeles que forman la región de la luz y se determinará si está dentro de los rangos de color de la luz (rojo, ámbar o verde) medidos en las muestras. El usuario también puede elegir si quiere combinar estos métodos de reconocimiento siendo las opciones posición-tono o *matching*-tono.

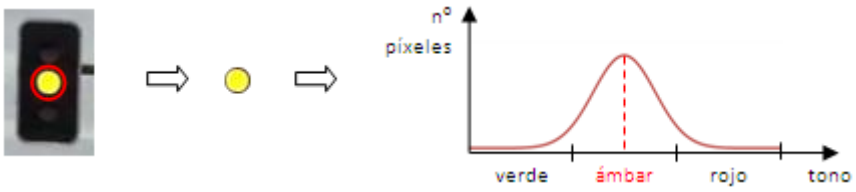


Figura 3.69: *Reconocimiento del color de la luz por tono.*

Finalmente la aplicación muestra por pantalla una ventana en la que se irán viendo las imágenes captadas junto con los resultados de la detección y del reconocimiento. Sobre cada imagen se dibujarán unos rectángulos y unas circunferencias que marcarán la ubicación, respectivamente, de la unidad del semáforo y de la luz. En función del color de la luz que se obtenga, debajo de las imágenes aparecerán representados los semáforos que se han detectado junto con su correspondiente señalización.

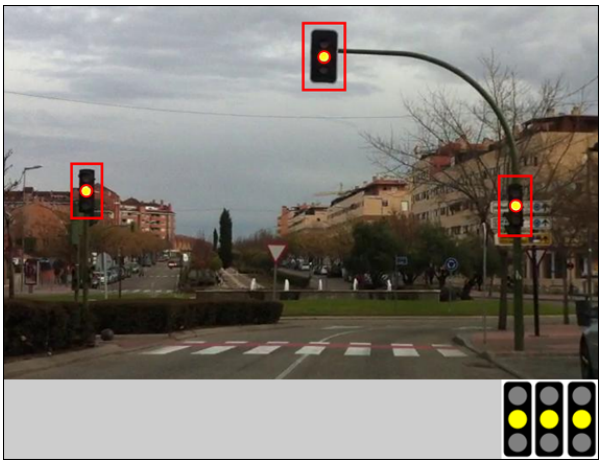


Figura 3.70: *Visualización de los resultados de la detección y el reconocimiento.*

3.3.3. Funciones Principales

Para conseguir el funcionamiento descrito, la aplicación *tldr* se compone de un conjunto de funciones en cada una de las cuales se realiza una tarea específica. A continuación, se intentará dar una descripción de la secuencia de tareas que se realizan en la aplicación *tldr* y las funciones que se utilizan para cada una de ellas:

- **Definir variables.** Lo primero que se hace en el programa es definir las variables que almacenarán los valores de los parámetros que se pasan como argumento de dicha aplicación. Como el número de parámetros es elevado, se decidió agrupar estas variables

en estructuras de datos, teniéndose una por cada tipo de función interna que se tiene. Así pues, se tienen estructuras para las fuentes de entrada de imágenes llamada `siParams` (*source input parameters)*, la detección de semáforos llamada `tldParams` (*traffic light detection parameters*), la detección de la luz llamada `sldParams` (*spot light detection parameters*), el reconocimiento de la luz llamada `slrParams` (*spot light recognition parameters*) y una última con los parámetros para mostrar los resultados de la detección y el reconocimiento llamada como `drdParams` (*detection and recognition display parameters*).

- **Inicializar parámetros.** Una vez definidas todas las variables de la aplicación, lo que se hace es llamar a la función `initializeParameters()` que, como su nombre indica, se encarga de inicializar las variables de los parámetros asignándoles unos valores por defecto. Esto se hace para asegurar que las variables tienen unos valores conocidos que llevan al programa a un estado estable.

- **Leer parámetros.** El siguiente paso es leer los parámetros introducidos como argumentos de la aplicación y se hace mediante la función `readParameters()`. Para ello se le pasan a la función las estructuras con las variables de los parámetros.

- **Imprimir ayuda.** Si se ha introducido el parámetro de ayuda, se imprimirá por pantalla la relación de parámetros que tiene la aplicación, qué significado tiene cada uno de ellos, y como utilizarlos. Después se saldrá de la aplicación.

- **Imprimir parámetros.** Para revisar los valores de los parámetros pasados a la aplicación, estos se imprimen por pantalla mediante la función `printParameters()`. Esto es útil para asegurarse de que los valores de los parámetros son los que se pretendía.

- **Comprobar parámetros.** Con la función `checkParameters()` se comprueba que los valores introducidos para los parámetros son correctos. Dependiendo del tipo de parámetro, se comprueba o bien que se haya introducido un valor, o que el valor esté dentro de unos límites, o que el valor pertenezca a una de las opciones permitidas. En caso de que el valor de algún parámetro sea incorrecto, se imprime por pantalla un mensaje error indicando el parámetro en el que se ha producido y que valor de éste se ha introducido. A continuación, se sale de la aplicación.

- **Cargar archivos.** Antes de comenzar a ejecutar las funciones principales de la aplicación, habrá que cargar los archivos externos con los que la aplicación trabaja. Esto se hace con la función `loadFiles()`. El primer archivo que solicita la aplicación es la fuente de entrada de imágenes, que bien puede ser o un archivo de texto con la lista de imágenes, o un archivo de video o una cámara. También necesita el archivo *.xml* que contiene al clasificador con el que trabajará el sistema detector. Para realizar la operación de *matching*, la aplicación requiere las tres plantillas con las que se evalúa el color de la luz. Por último, se cargan las imágenes que se utilizan en la pantalla de visualización para representar la señalización de los semáforos detectados. Si se produce algún error en la carga de estos archivos, se imprime un mensaje de error por pantalla con el tipo y nombre del archivo que no se ha podido cargar y se sale de la aplicación.

Con todos los parámetros y archivos listos, la aplicación entra en un bucle infinito en el que se va capturando imágenes de la fuente de entrada y se va aplicando sobre ellas las funciones de detección y reconocimiento. Dependiendo de la fuente de entrada, la programación para adquirir las imágenes es distinta, por lo que la aplicación deberá comprobar cuál es la fuente de entrada elegida por el usuario y en consecuencia seguir un hilo de ejecución u otro.

- **Cargar/Capturar imagen.** Ya dentro del bucle, lo primero que hace la aplicación es leer las imágenes de las fuentes de entrada. Si la fuente de entrada es un archivo de texto con la lista de imágenes a cargar, en cada iteración del bucle se lee una nueva línea de este archivo donde está la ruta de la siguiente imagen a procesar y con ella se carga dicha imagen (`inputImage`). Si la fuente de entrada es un archivo de vídeo o una cámara, en cada iteración del bucle se captura un fotograma, se comprueba si es múltiplo de una tasa de captura, y si lo es, se guarda como una imagen (`inputImage`) y se pasa a procesarla. Esta tasa de captura la elige el usuario e indica cada cuantos fotogramas del archivo de vídeo o de la cámara se captura uno.

- **Detección de semáforos.** Ya dentro del bucle, la primera tarea que se realiza es la de detectar los semáforos sobre la imagen. Para ello se utiliza la función denominada como `trafficLightsDetection()`. En esta función se implementa un algoritmo detector de semáforos que se ayuda del clasificador entrenado para ubicar las ventanas de la imagen donde se encuentran los semáforos. Como entrada a la función se necesita de la imagen a procesar (`inputImage`) y como salida se tiene un vector de rectángulos (`boundingBoxes`) que indican las coordenadas y el tamaño de las ventanas que contienen a los semáforos detectados sobre la imagen.

- **Detección de luz del semáforo.** Teniendo las ventanas de los semáforos localizadas, la aplicación llama a la función `spotLightsDetection()`. En esta función lo que se hace es buscar dentro de esas ventanas la región circular que representa a la luz. Para ello se utilizan una serie de técnicas morfológicas. Así pues, a la salida de esta función se tiene un vector de circunferencias (`boundingBlobs`) que indican la posición y el tamaño de las luces para cada semáforo.

- **Reconocimiento de luz del semáforo.** La siguiente función a la que llama la aplicación es `spotLightsRcognition()`. Esta función es la encargada de determinar la señalización que emiten los semáforos detectados. Para ello requiere de la imagen a procesar (`inputImage`) y de los `vectores` las referencias tanto de las ventanas de los semáforos (`boundingBoxes`) como de las regiones circulares de las luces (`boundingBlobs`). Dependiendo del tipo de reconocimiento que haya elegido el usuario, esta función llama a su vez a otras que se encargan de un tipo de reconocimiento específico. Así, se tienen tres funciones más para reconocer el color de las luces: por la posición (`spotLightsRcognitionByPosition()`), por la técnica de *matching templates* (`spotLightsRcognitionByMatching()`) o por el tono `spotLightsRcognitionByHue()`. Esta función también se encarga de combinar estas técnicas si el usuario lo elije. Como salida a esta función se tiene un vector (`spotLightsColor`) que indica mediante un código de números enteros, la señalización reconocida para los semáforos detectados

- **Visualizar detección y reconocimiento.** Para terminar, todos los resultados obtenidos de las funciones anteriores se utilizan para mostrarlos sobre una ventana con la función `detectionAndRecognitionDisplay()`. En dicha ventana se irá mostrando la imagen captada en cada iteración del bucle (`inputImage`), y sobre ella se dibujarán los rectángulos (`boundingBoxes`) y circunferencias (`boundingBlobs`) que localizan a los semáforos y a las luces. Debajo de esta imagen se representarán unas figuras de los semáforos detectados con la señalización que se ha reconocido (`spotLightsColor`).

Llegada aquí, la aplicación ejecuta otra iteración del bucle y otra imagen a procesar. Para salir de este bucle, bastaría con presionar la tecla `escape` sobre la ventana de visualización.

3.3.4. Detección de Semáforos

3.3.4.1. Introducción

Aquí se explicará más detalladamente el proceso de detección de los semáforos que se realiza en la aplicación. Como ya se comentó, este proceso consiste en aplicar sobre las imágenes un detector que utiliza el clasificador de semáforos entrenado para localizar las regiones donde se encuentran los semáforos. Esta tarea se realiza mediante la función `trafficLightsDetection()`, a la que se llama de la siguiente forma:

```
trafficLightsDetection( inputImage, boundingBoxes, tldParams )
```

Como se puede apreciar en la llamada a la función, ésta solicita que se la pasen tres parámetros. El primero llamado `inputImage` es la matriz que contiene a la imagen a procesar. Después se tiene un vector llamado `boundingBoxes` en el que se almacenarán las marcas rectangulares que van a ubicar a los semáforos detectados en la imagen. Por último se tiene una estructura llamada `tldParams` que contiene el resto de parámetros con los que trabaja.

3.3.4.2. Preprocesado de las Imágenes

La primera tarea que se realiza en esta función es la de preprocesar la imagen de entrada (`inputImage`) para adaptarla y mejorar el proceso de detección. Como el detector trabaja en escala de grises, la primera operación que se realiza es convertir la imagen de entrada de escala de color a dicha escala de grises.

Figura 3.71: *Convertir la imagen a escala de grises.*

Después de tener la imagen en escala de grises, lo que se hace es reducir el tamaño de la misma con el objetivo de aumentar la velocidad de procesamiento del algoritmo detector. Para ello se utiliza uno de los parámetros se pasan a la función y que se llama `tldParams.shrinkFactor`. Este factor indica el porcentaje en el que es reducida la imagen de entrada.

Figura 3.72: *Reducir la imagen.*

La siguiente operación de preprocesado, tiene como objetivo enfocar la detección en las zonas de la imagen donde es más frecuente que se puedan encontrar los semáforos. En este caso, lo que se hace es recortar horizontalmente la imagen por la parte de abajo ya que los semáforos se suelen encontrar en las zonas superiores (ver ***Figura 3.73***). Para establecer el tamaño del recorte, se utiliza el parámetro `tldParams.cropFactor` que se multiplica por la altura de la imagen para obtener la distancia desde abajo a la que se recorta la imagen. Esto además sirve para mejorar el tiempo de procesamiento del detector ya que la imagen es de menor tamaño. También sirve para evitar que se detecten falsos semáforos como por ejemplo los faros de los vehículos.

Figura 3.73: *Recortar la imagen.*

La última operación de preprocesado, consiste en filtrar la imagen mediante un filtro *gaussiano* con el objetivo de eliminar el ruido que pueda contener. Esto hace que la imagen parezca desenfocada pero se sigue manteniendo la silueta de los semáforos ahora libres de ruido. Para elegir el tamaño del filtro *gaussiano* se utiliza el parámetro `tldParams.blurSize`. En la imagen de la ***Figura 10*** se puede ver el resultado de aplicar el filtro.

Figura 3.74: *Filtrar la imagen.*

3.3.4.3. Detección de los Semáforos

El siguiente paso que se realiza es detectar los semáforos sobre la imagen preprocesada. Para ello, se utiliza una función especial que proporcionan las librerías de *OpenCV* y que se llama `detectMultiScale()` **[31]**. Internamente esta función está formada por un algoritmo detector que trabaja en torno al clasificador entrenado. La primera parte de este algoritmo consiste en ir reduciendo la imagen iterativamente mediante un factor de escala (`tldParams.scaleFactor`) para así obtener los semáforos en varios tamaños y que se tengan más posibilidades de encontrarlos. Después, se van extrayendo ventanas con un tamaño específico (`tldParams.minSize` y `tldParams.maxSize`) de cada una de las imágenes escaladas y se las pasa al clasificador para que las evalúe. Para aquellas ventanas en las que el clasificador haya encontrado un semáforo, se obtienen sus localizaciones y sus tamaños, y éstos se guardan en un vector pasado como argumento a la función (`boundingBoxes`). Con este vector se tendrán ubicados a los semáforos en la imagen. Para un mejor entendimiento de lo explicado, en la siguiente imagen se tiene un esquema del funcionamiento del detector con todos los pasos que ejecuta:

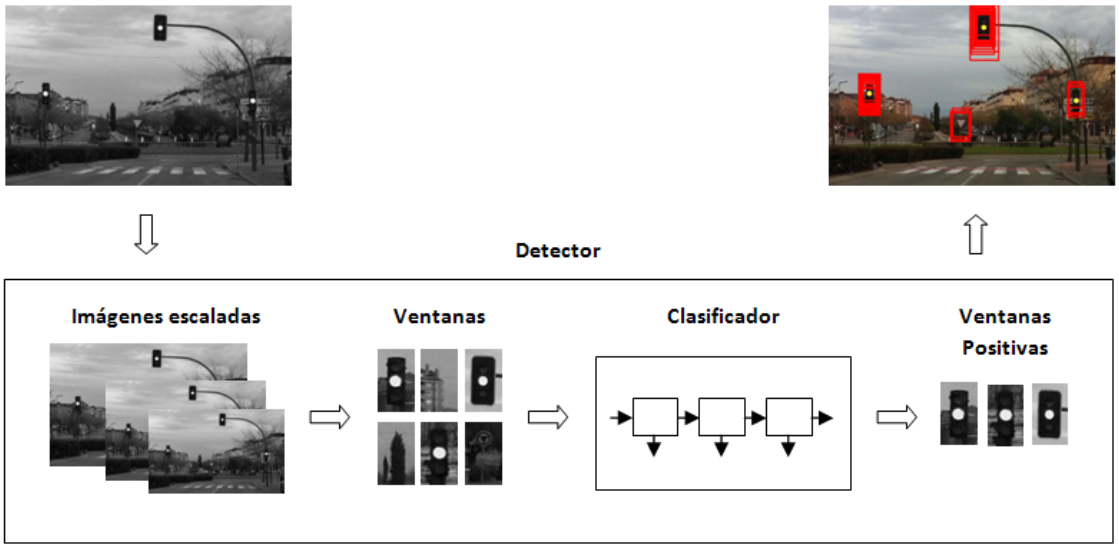


Figura 3.75: *Esquema de funcionamiento del detector.*

3.3.4.4. Filtrado de las Marcas de Detección.

De la operación anterior se obtiene un vector de datos con las coordenadas y tamaño de las ventanas de la imagen que contienen a los semáforos. Para que sea más simple nombrar a estos datos se les llamará a partir de ahora como marcas de detección. Pues bien, ocurre que, por las características del detector, para un mismo semáforo se obtienen varias de estas marcas de detección e idealmente lo que se pretende es obtener una única. Para conseguir este objetivo, lo que se hace es someter a estas marcas a un proceso de filtrado con el fin de obtener aquella que mejor represente la ventana del semáforo.

Filtrar marcas por agrupación. La primera operación de filtrado que se realiza, consiste en agrupar bajo una única marca aquellas que tienen semejante localización y tamaño. La marca rectangular final es la media de las marcas que forman el grupo. Esta operación se realiza mediante la función de *OpenCV* llamada `groupRectangles()` **[31]**. Esta función trabaja con el vector de marcas (`boundingBoxes`) y un parámetro (`tldParams.minGroup`) que indica el mínimo número de vecinos que debe tener una marca para que no se excluya de la agrupación y no se elimine. También se le pasa como parámetro un vector (`tldParams.bboxesScores`) donde se guarda el número de marcas con las que se han formado las marcas resultantes. Este vector servirá como puntuación para valorar que marcas son las que más probabilidades tienen de representar a un semáforo.

Figura 3.76: *Filtrado de marcas por agrupación.*

Filtrar marcas por *non-maximun suppression*. Como se puede ver en la ***Figura 3.77***, puede ser que la técnica de agrupación no sea suficiente para combinar todas las marcas vecinas bajo una única. Para ello, a continuación se utiliza otra técnica de filtrado de marcas más sencilla y llamada *non-maximun suppression*. Esta técnica consiste en evaluar el nivel de solapamiento que hay entre dos marcas y si supera un umbral establecido (`tldParams.minOverlap`) eliminar aquella que tenga menor puntuación. El nivel de solapamiento de dos marcas se calcula como el área de la intersección entre el área de unión.

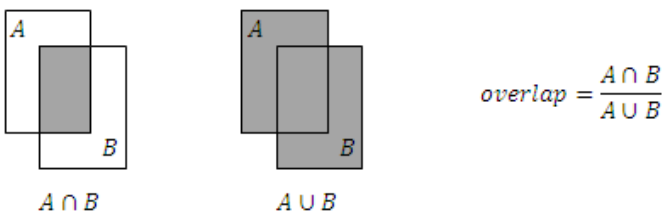


Figura 3.77: *Área de solapamiento.*

En la siguiente imagen se puede observar el resultado de aplicar *non-maximun suppression*:

Figura 3.78: *Filtrado de marcas por non-maximun suppression.*

Filtrar marcas por número máximo. En la última operación de filtrado de marcas, se escogen de mayor a menor puntuación (`tldParams.bboxesScores`) solamente un **número** determinado de ellas que vendrá fijado por el parámetro `tldParams.maxNumber`. Con esta operación se asegura que no se detectan más semáforos de los que puedan aparecer en las imágenes y así eliminar posibles falsos positivos. En la ***Figura 3.79*** se puede apreciar el efecto de esta operación, donde se ha considerado que el máximo número de semáforos que pueden aparecer es de tres y por tanto la cuarta marca que se ha registrado debe pertenecer a un falso positivo.

Figura 3.79: *Filtrado de marcas por número máximo.*

En este punto, la función `trafficLightsDetection()` termina y deja como resultado un vector que contiene las marcas de detección con las que ubicar las regiones donde se encuentran los semáforos dentro de la imagen. Este vector será utilizado después para localizar la luz del semáforo.

3.3.5. Detección de la Luz de los Semáforos

3.3.5.1. Introducción

Otra de las funciones principales de la aplicación *tldr* es la que se encarga de detectar la región donde se encuentra la luz del semáforo. Esto se hace para después hacer un reconocimiento del tipo de luz que está emitiendo el semáforo y así saber su correspondiente señalización. La función que realiza esta operación se llama `spotLightsDetection()` y se la ejecuta de la siguiente manera:

```
spotLightsDetection( inputImage, boundingBoxes, boundingBlobs, sldParams )
```

Entre sus parámetros se encuentra la imagen a procesar (`inputImage`), las marcas de de los semáforos en dicha imagen (`boundingBoxes`), un vector de datos donde se guardarán las marcas de detección que ubican a las luces en los semáforos (`boundingBlobs`) y , por último, una estructura de datos (`sldParams`) que contiene los parámetros internos con los que trabaja está función. Para entender cómo trabaja esta función, a continuación se hará una explicación detallada de cada uno de los pasos que se siguen para localizar la luz del semáforo.

3.3.5.2. Extracción de las Ventanas de los Semáforos

La primera tarea que se realiza en la función `spotLightsDetection()` consiste en extraer iterativamente las ventanas de la imagen en las que se encuentran los semáforos. Para ello, se utilizan las marcas de detección (`boundingBoxes`) obtenidas anteriormente y que indican las coordenadas y el tamaño de esas ventanas.

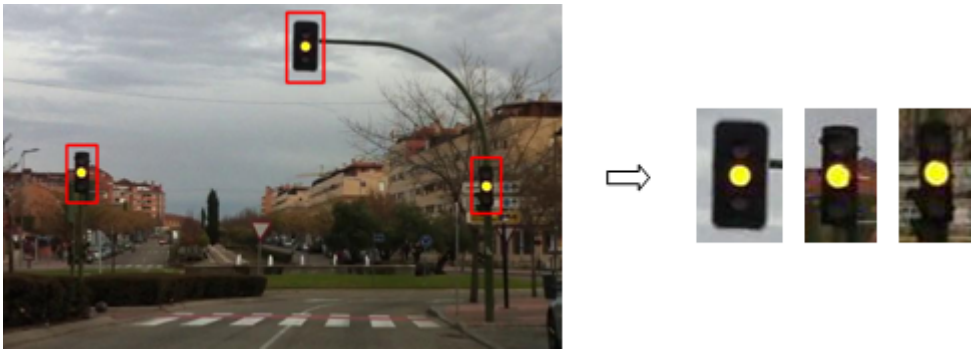


Figura 3.80: *Extracción de las ventanas de los semáforos.*

3.3.5.3. Preprocesado de las Ventanas de los Semáforos

Para cada una de las ventanas extraídas, lo siguiente que se hace es someterlas a un pequeño preprocesado con el fin de adaptarlas para las siguientes operaciones. La primera operación que se hace sobre la ventana del semáforo es convertirla de escala de color a escala de grises. Esta conversión se hace para poder utilizar después otras técnicas que trabajan con este formato. Después lo que se hace es filtrar la ventana con un filtro *gaussiano* de tamaño `sldParams.blurSize` para reducir el ruido. Por último, se normaliza la ventana para que la intensidad de sus píxeles ocupe todo el rango de valores posibles y así resaltar el foco de luz del resto de la unidad del semáforo.

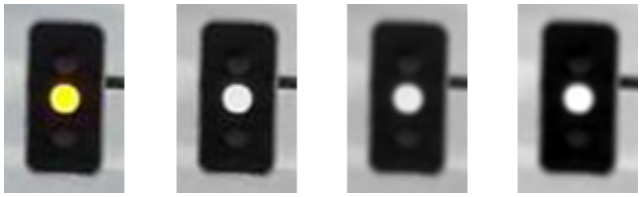


Figura 3.81: *Preprocesado de la ventana del semáforo.*

3.3.5.4. Detección de la Luz de los Semáforos

Con la ventana del semáforo preprocesada se pasa a detectar la región de la luz. Para realizar esta tarea se ha diseñado específicamente una función a la que se ha llamado como `blobsDetector()`. La forma de llamar a esta función es la siguiente:

```
blobsDetector( normalizedRoi, detectedBlobs, sldParams.bdParams )
```

donde se solicita la ventana que contiene al semáforo una vez preprocesada (`filteredRoi`), un vector para guardar las marcas de detección (`detectedBlobs`) y los parámetros internos con los que trabaja esta función (`sldParams.bdParams`). Una vez dentro de esta función se realizan las siguientes operaciones:

Umbralización progresiva. Consiste ir aplicando de forma incremental (`bdParams.threshStep`) un umbral sobre la ventana del semáforo de tal forma que para cada umbral se obtiene una versión binaria de la ventana. El rango en el que se mueve el umbral viene determinado por los parámetros `bdParams.minThresh` y `bdParams.maxThresh`. Esto se hace para encontrar la región que representa el foco de luz.

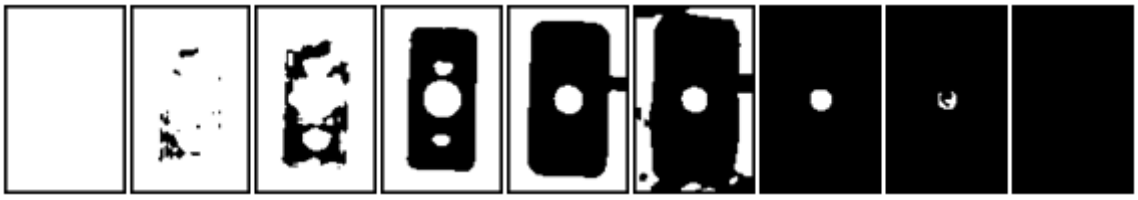


Figura 3.82: *Umbralización progresiva.*

Buscar regiones. Para cada versión binaria de la ventana, se aplica la función de *OpenCV* `findContours()` **[31]** que busca las regiones cerradas presentes en dicha ventana y las guarda en un vector de datos.

Una vez encontradas las regiones, lo que se hace es buscar aquellas que representan la luz del semáforo. Para ello, se somete a dichas regiones a un proceso de filtrado dejando pasar aquellas que cumplan una serie de condiciones. Como la región del foco de luz va a ser circular, estas condiciones están relacionadas con un conjunto de indicadores que determinan la circularidad de las regiones. Para calcular estos indicadores será necesario obtener primero los momentos de las regiones.

Filtrar regiones por diámetro. El primer filtro consiste en comprobar que el área de la región se aproxima a la que tiene el foco de luz. Como el área del foco de luz va a ser circular, el área se mide en términos del diámetro. Para ello será necesario haber introducido el margen de diámetros entre los que se mueve el foco de luz que son `bdParams.minDiam` y `bdParams.maxDiam`.

$$area = \frac{\pi\ diameter^2}{4} \tag{3.1}$$

Filtrar regiones por circularidad. Aquí se calcula la circularidad de la región y se compara con la que se espera tener para el foco de luz. Si la región está dentro de los límites se continúa si no se desecha y se pasa a evaluar la siguiente. El margen de valores de la circularidad de la luz a detectar se establece con `bdParams.minCirc` y `bdParams.maxCirc`.

$$circularity = \frac{4\ \pi\ area}{perimeter^2} \tag{3.2}$$

Filtrar regiones por inercia. En el siguiente tipo de filtro se comprueba que la relación de inercia de la región está dentro de los límites establecidos (`bdParams.minInert` y `bdParams.maxInert`).

Filtrar regiones por convexidad. Con este tipo de filtro se buscan aquellas regiones cuya convexidad esté entre un determinado rango de valores. La convexidad se calcula como la relación entre el área de la región entre el área del cerco convexo de la misma región. Para seleccionar el rango de valores aceptables se utilizan `bdParams.minConvex` y `bdParams.maxConvex`.

$$convexity = \frac{area}{area\ cerco\ convexo} \tag{3.3}$$

Terminada la operación de filtrar, se calcula el centro y el diámetro de las regiones que han pasado, estas referencias se guardan en el vector `detectedBlobs` como marcas de detección y se pasa a analizar una nueva versión binaria de la ventana.

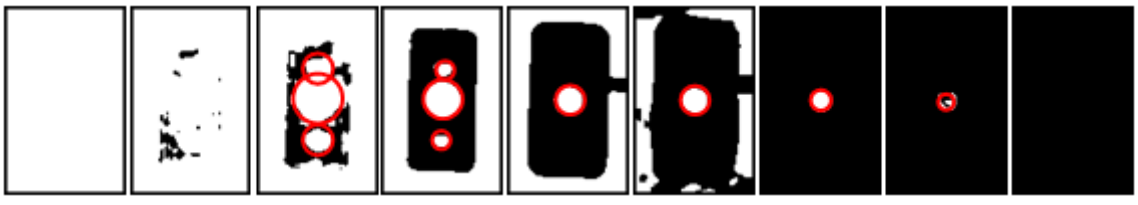


Figura 3.83: *Detección de regiones circulares.*

Al terminar la función `blobsDetector()`, se tendrá en el vector `detectedBlobs` todas las marcas de detección que indican las posible posiciones y tamaños del foco de luz dentro de la ventana del semáforo.

Figura 3.84: *Resultado de la detección de la luz del semáforo.*

3.3.5.5. Filtrado de las Marcas de Detección

El problema que aparece después de detectar el foco de luz es que se tienen varias marcas de detección candidatas cuando solo se tiene que tener una. Este inconveniente es el mismo que se encontró en la detección de los semáforos por lo que se seguirá la misma metodología. Hay que comentar que aunque las marcas se representan con forma circular en realidad se trabaja con formas cuadradas por lo que se pueden utilizar las mismas funciones que en la detección de semáforos.

Filtrar marcas por agrupación. Los primero que se hace es aplicar sobre las marcas de detección la función `groupRectangles()` **[31]**. Como ya se comentó, esta función agrupa aquellas marcas que tiene localizaciones y tamaños similares, y las combina obteniendo una marca promedio de cada grupo (`detectedBlobs`). Para que una marca se considere en esta operación deberá tener al menos el número de vecinos indicado por el parámetro `sldParams.minGroup`. Además, esta función guarda en un vector (`blobScores`) el número de marcas con las que se ha formado la marca resultante, dato que se utilizará como puntuación para valorar la probabilidad de que una marca se corresponda realmente con la luz. Se entiende que cuantas más marcas haya alrededor de un mismo punto, mayor es la probabilidad de que en ese punto se encuentre el objetivo buscado.

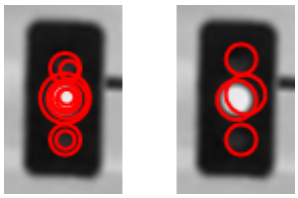


Figura 3.85: *Filtrado de marcas por agrupación.*

Filtrar marcas por *non-maximun suppression*. Después, se aplica la técnica de *non-maximun suppression* para eliminar aquellas marcas cuyo nivel de solapamiento está por encima de un determinado valor (`sldParams.minOverlap`). Entre las dos marcas solapadas se elimina aquella que tiene menor puntuación (`blobScores`).Como ya se explicó, el solapamiento se calcula como el área de intersección entre el área de unión de dos marcas.

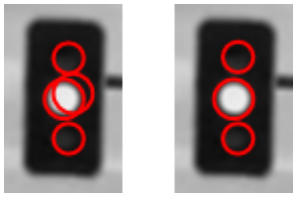


Figura 3.86: *Filtrado de marcas por non-maximun suppression.*

Filtrar marcas por número máximo. La última operación de filtrado de marcas consiste en escoger aquella marca de detección que tenga mayor puntuación. Esto es así, porque en la ventana del semáforo se espera tener un único foco de luz.

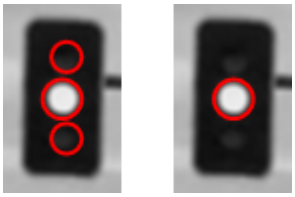


Figura 3.87: *Filtrado de marcas por número máximo.*

Llegado aquí, el proceso de detección de la luz se repite para una nueva ventana de semáforo hasta haber procesado todas. Particularmente, se puede dar la situación de que en alguna ventana no se obtenga ninguna marca de detección de la luz, siendo en ese caso eliminada directamente la marca de detección que hace referencia a esta ventana por considerarse como un falso positivo. Finalmente, al terminar la función `spotLightsDetection()` se obtiene como resultado un vector (`boundingBlobs`) que contiene las marcas de detección con las que ubicar la luz de cada uno de los semáforos encontrados.

3.3.6. Reconocimiento de la Luz de los Semáforos

3.3.6.1. Introducción

Hasta ahora se ha explicado como detectar las ventanas en las que se encuentran los semáforos y ubicar el foco de luz dentro de ellas. Esta información será utilizada para realizar un reconocimiento de la señalización que muestran los semáforos captados. Para ello, la aplicación *tldr* se vale de la función `spotLightsRecognition()`:

```
spotLightsRecognition( inputImage, boundingBoxes, boundingBlobs,
                       spotLightColors, slrParams )
```

Esta función solicita como parámetros la imagen captada (`inputImage`), los vectores de datos que contienen las marcas de detección de los semáforos y de las luces (`boundingBoxes` y `boundingBlobs`) y, por último, los parámetros con los que trabajan las funciones internas a ésta (`slrParams`).

En verdad esta función no realiza directamente ningún reconocimiento, tan solo se encarga de llamar a aquellas funciones que realizan el tipo de reconocimiento elegido por el usuario. Esta elección se realiza a través del parámetro `slrParams.recognType`. Ya se explicó que existen tres tipos de reconocimiento que son por posición de la luz, por *matching templates* y por el tono de color de la luz, y otros dos más que surgen como combinación de éstos que son posición-tono y *matching*-tono. Para entender cómo funcionan estos tipos de reconocimiento, a continuación se hará una descripción detallada de cada una de ellos.

3.3.6.2. Reconocimiento de la Luz por Posición

El primer tipo de reconocimiento de la luz que se ha diseñado consiste en ubicar la posición de la luz respecto de la unidad del semáforo. Como ya se sabe, el color de la luz se puede identificar a partir de la posición de ésta teniéndose luz roja para la posición de arriba, luz ámbar para la posición del centro y luz verde para la posición de abajo. Básicamente con esta idea es con la que trabaja la función `spotLightsRecognitionByPosition()`, a la que se llama desde `spotLightsRecognition()` de la siguiente forma:

```
spotLightsRecognitionByPosition( inputImage, boundingBoxes, boundingBlobs,
                                 spotLightColors, slrParams.pParams )
```

Los parámetros son los mismos que se le pasaron a la función `spotLightsRecognition()` salvo que en este caso cambian los parámetros internos siendo los específicos de esta función (`slrParams.pParams`). Una vez conocidos los parámetros se pasará a describir las tareas que se realizan en esta función para identificar la luz del semáforo.

Extraer las ventanas de los semáforos. Al empezar, lo primero que se encuentra en esta función es un bucle iterativo en el que se van extrayendo, de la imagen captada (`inputImage`), una a una las ventanas de los semáforos a partir de las marcas de detección (`boundingBoxes`). Después de extraer una de las ventanas se la somete al proceso de reconocimiento.

Preprocesar la ventana del semáforo. La primera parte del proceso de reconocimiento consiste en preprocesar la ventana del semáforo. La secuencia de preprocesado consiste en convertir la ventana a escala de grises, eliminar el ruido con un filtro *gaussiano* y por último normalizar la imagen para mejorar el contraste.

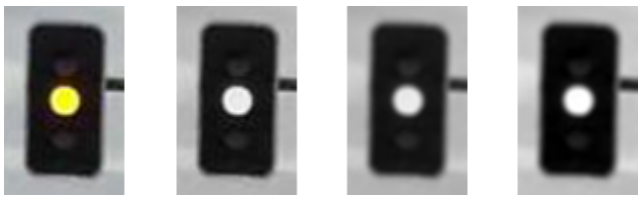


Figura 3.88: *Preprocesado de la ventana del semáforo.*

Detectar bordes horizontales. Como el objetivo es detectar los bordes superior e inferior del módulo del semáforo para posicionar correctamente la luz, se van a obtener todos los bordes horizontales que se encuentren en la ventana del semáforo para después buscar aquellos que pertenezcan al semáforo. Para ello se aplica a la ventana el operador horizontal de *Sobel* (de tamaño `pParams.sobelSize`) y después se binariza la ventana con un umbral (`pParams.edgesThresh`) para obtener los bordes más importantes.

Figura 3.89: *Detección de bordes horizontales con el operador de Sobel.*

Detectar bordes del semáforo. Una vez que se tienen los bordes horizontales que se encuentran en la ventana, lo que se hace es buscar aquellos que pertenecen al semáforo. Para reducir las posibilidades, se definen unas regiones de búsqueda concretas donde se sabe que se pueden encontrar dichos bordes. La primera región va desde el límite superior de la luz del semáforo hasta el alto de la ventana considerando un ancho igual al diámetro la luz. La segunda región va desde el límite inferior de la luz hasta el límite inferior de la ventana con un ancho igual al del diámetro de la luz. Este diámetro se calcula como el ancho de la marca de detección de la luz (`boundingBlobs[i].width`) multiplicado por un factor llamado `pParams.boundFactor` que se utiliza para aumentar el valor del diámetro que se considera y así evitar que los bordes horizontales de la propia luz caigan dentro de las regiones de búsqueda.

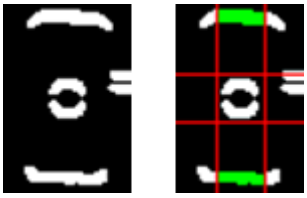


Figura 3.90: *Definir las regiones de búsqueda.*

Únicamente considerando los bordes que se encuentran dentro de estas regiones, se calcula el histograma lateral de la ventana donde se reflejará la posición vertical de los bordes encontrados.

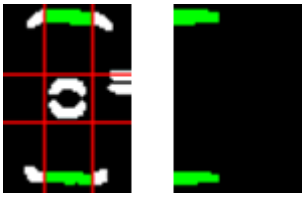


Figura 3.91: *Calcular el histograma lateral.*

Sobre el histograma lateral se aplica un umbral para eliminar aquellos bordes que no tienen un ancho suficiente. El ancho que se considera de referencia es el de la marca de detección de la luz (`boundingBlobs[i].width`) multiplicado por un factor llamado `pParams.widthFactor.`

Figura 3.92: *Eliminar los bordes no suficientemente anchos.*

Finalmente, se supone que los bordes que han pasado el umbral pertenecen a los del semáforo. Como va a haber varias posiciones para identificar al mismo borde, se cogerán aquellas más internas a dicho borde. Si en este proceso no se encuentra ningún borde se considerarán como tal los bordes de la propia ventana.

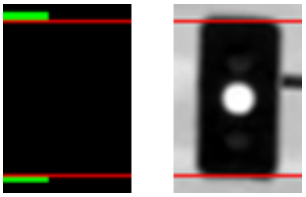


Figura 3.93: *Obtener las posiciones de los bordes.*

Identificar el color de la luz. Una vez que se tienen localizados los bordes superior e inferior del semáforo ya se puede identificar la posición de la luz. Para ello se divide la altura del semáforo en tres regiones, de tal manera que si la luz está en el tercio superior el semáforo estará en rojo, si está en el tercio central estará en ámbar y si lo está en el tercio inferior estará en verde.

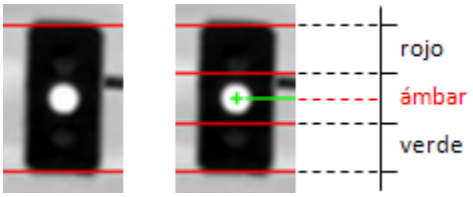


Figura 3.94: *Identificar la posición de la luz del semáforo.*

En ocasiones sucede que el detector de semáforos no consigue cuadrar bien la ventana de detección en torno a éste debido a que el fondo de la imagen tiene un tono tan oscuro que se confunde con el propio fondo del semáforo. Al darse este caso se tendrá una ventana donde aparece el semáforo entrecortado, por lo que la idea de dividir la altura del semáforo en tres partes y ubicar la luz en alguna de ellas ya no sería válido. Ante esta situación lo que se hace es comprobar la distancia entre los bordes del semáforo detectado. Aclarar que como el semáforo aparece entrecortado alguno de los bordes de éste no podrá ser localizado y por tanto se le asignan los de la ventana. Entonces la distancia entre los bordes del semáforo se compara con un valor que se obtiene de multiplicar el diámetro de la luz (que se corresponde con el ancho de la marca de detección `boundingBlobs[i].width`) con un factor denominado como `pParams.distFactor`. Este factor representa la relación de tamaño que existe entre la distancia que separa dos lámparas de un mismo semáforo y el diámetro de esas lámparas. Así pues, se pueden dar tres situaciones en la comparación de la distancia entre los bordes con la distancia entre las lámparas:

- Si es dos veces mayor, se hace una división en tres regiones: rojo, ámbar, verde.

- Si es menor que dos y mayor que una vez, se hace una división en dos regiones, siendo estas regiones rojo y ámbar si se detecta el borde superior o ámbar y verde si se detecta el borde inferior.

- Si la distancia es menor que una vez, se considera una única región siendo ésta rojo si se detecta el borde superior o verde si se detecta el borde inferior

En la siguiente imagen se muestra el modo de operar cuando la ventana de detección está descuadrada.

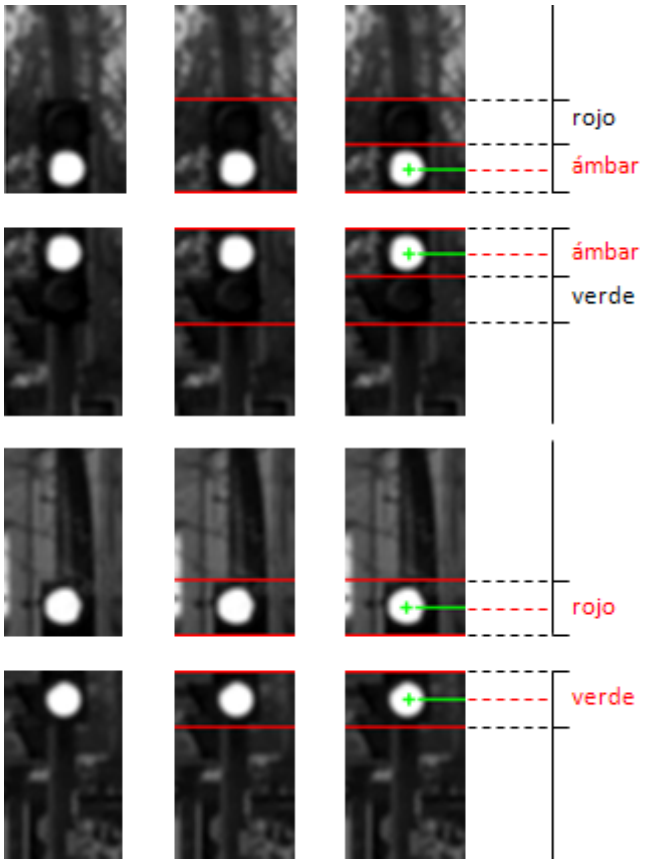


Figura 3.95: *Identificar el color de la luz con la ventana de detección descuadrada.*

Finalmente, a la luz del semáforo se le asigna un número que indica el color detectado y éste se guarda en el vector de datos `spotLightColors`. Hay que comentar que existe la posibilidad de que no se pueda identificar el color de la luz, en cuyo caso se le etiqueta como desconocido. Una vez que se hayan evaluando todas las ventanas de los semáforos detectados en la imagen, la función `spotLightsRecognitionByPosition()` habrá terminado.

3.3.6.3. Reconocimiento de la Luz por *Matching*

Otro método que se ha diseñado para identificar la señalización de los semáforos es mediante *matching templates*. Esta técnica consiste en comparar el semáforo detectado con un conjunto de plantillas que representan a los semáforos en las distintas posiciones que puede tener la luz. Aquella plantilla que más coincida con la posición del semáforo será la que identifique el color de la luz. Esta idea es la que se implementa bajo la función llamada como `spotLightsRecognitionByMatching()`:

```
spotLightsRecognitionByMatching( inputImage, boundingBoxes, boundingBlobs,
                                 spotLightColors, slrParams.mParams )
```

Entre los parámetros que solicita esta función se encuentran la imagen a evaluar (`inputImage`), los vectores de datos con las marcas de detección de los semáforos y sus luces (`boundingBoxes` y `boundingBlobs`) y los parámetros internos específicos para este tipo de reconocimiento (`slrParams.mParams`). A partir de estos parámetros, la función realiza las operaciones que se explican a continuación.

Cargar y redimensionar las plantillas. Para aplicar *matching templates* a cada una de las ventanas lo primero que se necesita son las plantillas. Si se recuerda, son pasadas como parámetro a la aplicación *tldr* y se cargan al comienzo de la ejecución de ésta mediante la función *loadFiles()* sobre el parámetro `mParams.matchingTemplates`. En total se tienen tres tipos de plantillas que se identifican con cada una de las posiciones de la luz respecto del semáforo que son arriba (rojo), centro (ámbar) y abajo (verde).

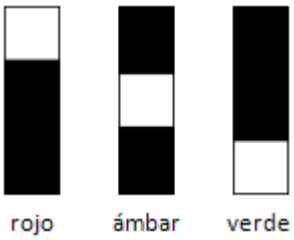


Figura 27: *Plantillas de los semáforos.*

Sabiendo donde se encuentran las plantillas, se puede comenzar a trabajar con ellas. La función `spotLightsRecognitionByMatching()` es un proceso bucle en el que cada iteración se procesa uno de los semáforos detectados en la imagen captada. El primer paso en este bucle consiste en leer el diámetro de la luz del semáforo que actualmente se está evaluando (`boundingBlobs[i].width` o `boundingBlobs[i].height`). Con este diámetro y el ancho de las plantillas se calcula un factor de redimensionado que se utilizará sobre las propias plantillas para hacerlas coincidir con el tamaño del semáforo. Hay que comentar que estas plantillas han sido construidas a partir de muestras de semáforos para que guarden las proporciones del tamaño de éstos.

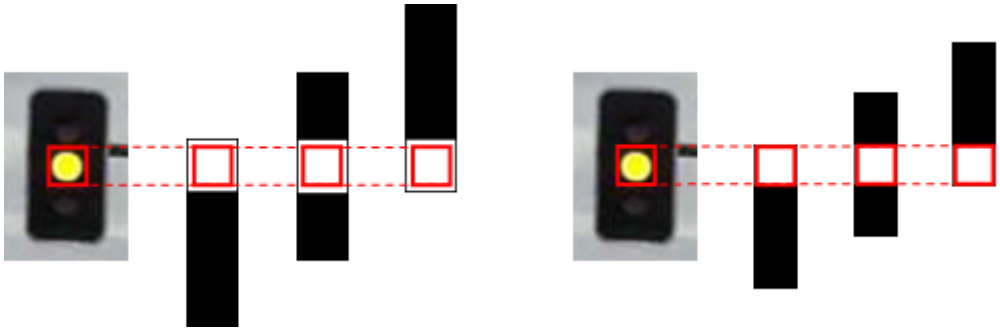


Figura 3.96: *Redimensionar las plantillas.*

Obtener y preprocesar las regiones de interés. Una vez que se tienen las plantillas adaptadas al tamaño del semáforo lo que se hace es obtener la región del semáforo donde se van a aplicar. Lo que se busca es que la marca de detección de la luz del semáforo coincida con la región de la plantilla que representa esa luz. Para ello se tiene, acompañando a las plantillas, otro parámetro que lo introduce el usuario y que se llama `mParams.lightBox` donde están la localización y el tamaño de la región que representa la luz en las plantillas. Aclarar que estas referencias también fueron redimensionadas de acuerdo con el nuevo tamaño de las plantillas. Haciendo coincidir la localización de las marcas de detección (`boundingBlobs[i].x` y `boundingBlobs[i].y`) con la región de la luz de las plantillas (`resizedLightBox[i].x` y `resizedLightBox[i].x`), se obtiene la región donde aplicarlas.

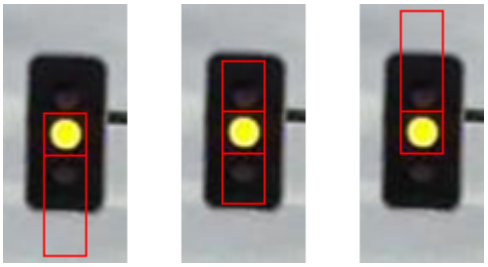


Figura 3.97: *Obtener las regiones de interés.*

Para mejorar los resultados del *matching*, se realiza un preprocesado sobre las regiones donde se van a aplicar las plantillas. Este proceso consiste en convertir la región a escala de grises, aplicar un filtro *gaussiano* y, por último, normalizar los niveles de intensidad.

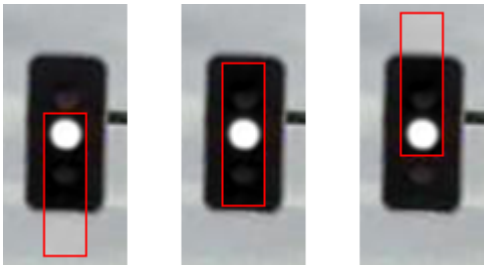


Figura 3.98: *Preprocesar las regiones de interés.*

Aplicar *matching*. Con la región de interés localizada y preprocesada se procede a realizar la operación de *matching* para cada una de las plantillas. Para ello, se utiliza la función `matchTemplate()` de *OpenCV* **[31]**. Esta función permite elegir entre varias formas de *matching*, siendo la correlación normalizada la elegida. Este tipo de *matching* la siguiente expresión:

$$M = \frac{\sum_{x,y}(T(x,y) \cdot R(x,y))}{\sqrt{\sum_{x,y} T(x,y)^2 \cdot \sum_{x,y} R(x,y)^2}} \tag{3.4}$$

Siendo M el nivel de coincidencia entre la plantilla T y la región de interés R. La operación se puede interpretar como sobreponer las plantillas en las regiones de interés y evaluar la proximidad entre los niveles de sus pixeles. Esta forma de verlo se intenta mostrar en la siguiente imagen:

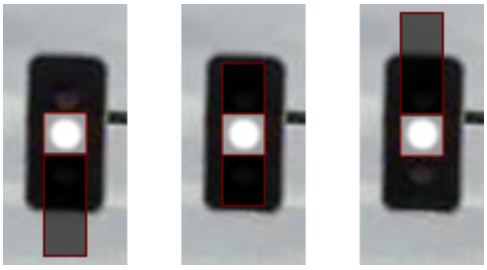


Figura 3.99: *Aplicar matching templates.*

Identificar el color de la luz. Una vez aplicado el *matching*, se comprobará que plantilla es la que mayor nivel de coincidencia ha obtenido y se asignará el color que representa dicha plantilla a la luz del semáforo (`spotLightColors[i]`).

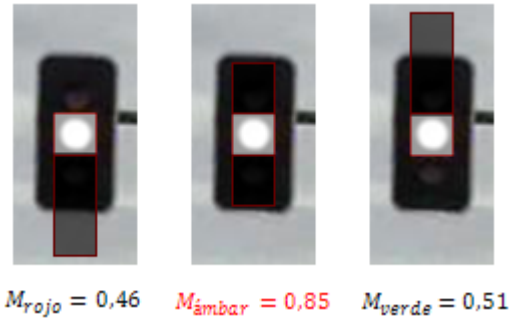


Figura 3.100: *Identificar el color de la luz.*

Los pasos descritos hasta aquí se repetirán para otro semáforo hasta haber identificado la señalización de todos los semáforos que contiene la imagen actual.

3.3.6.4. Reconocimiento de la Luz por Tono

La última forma de identificar la señalización de los semáforos es mediante el tono color que presenta la luz. Hasta ahora, los métodos vistos trabajan con un espacio de color en escala de grises donde identificar el color de la luz se basa en asociarlo con la posición que tiene ésta en el semáforo. Esta forma de operar es útil siempre que se trabaje con los semáforos convencionales de tres posiciones (rojo, ámbar, verde), pero la realidad es que existen más tipos de semáforos (como los de una o dos posiciones), o señalizaciones (como ámbar intermitente en la posición central e inferior de un semáforo de tres posiciones). Además, como lo único que permanece inmutable es el significado del color de la luz, sea el tipo de semáforo que sea, lo más lógico sería reconocer la señalización de un semáforo por el propio color de su luz. Así pues, esto es lo que se plantea en el siguiente y último método de reconocimiento de la luz de los semáforos. Dicho método está implementado bajo la función `spotLightsRecognitionByHue()`, la cual tiene el siguiente prototipo de llamada:

```
spotLightsRecognitionByHue( inputImage, boundingBoxes, boundingBlobs,
                            spotLightColors, slrParams.hParams )
```

Al igual que para los otras funciones de reconocimiento, esta función requiere de la imagen a evaluar (`inputImage`), de las marcas de detección tanto de los semáforos como de sus luces (`boundingBoxes` y `boundingBlobs`) y de los parámetros de las funciones internas (`slrParams.hParams`). Dentro de la función, las operaciones que se realizan son las que a continuación se describen.

Construir una máscara de la luz. Internamente la función es un bucle en el que en cada iteración se extrae una ventana de semáforos de los detectados en la imagen. La primera operación que se realiza sobre cada de estas ventanas es construir una máscara para evaluar solo los pixeles que componen la luz del semáforo. Esta máscara tiene forma de corona circular por dos razones. La primera razón es debida a que se produce un efecto halo cuando la cámara capta la luz del semáforo. Esto hace que la componente de color de la luz forme un anillo en torno a la lámpara del semáforo en vez de estar uniformemente distribuida. La otra razón es porque así se evita el efecto de deslumbramiento que se produce en la cámara. Este efecto se suele concentrar en la zona central de la lámpara por ser la de mayor intensidad y hace que la componente de color de esa zona no se pueda identificar. Para construir la máscara se utilizan dos parámetros que son introducidos por el usuario y que son `hParams.intHaloFactor` y `hParams.extHaloFactor`. Estos parámetros son multiplicados por el diámetro de la luz detectada (`boundingBlobs[i].width`) para obtener los diámetros interior y exterior de lo que será la corona de la máscara.

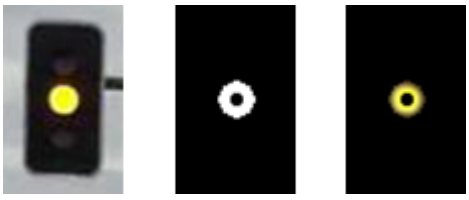


Figura 3.101: *Construir una máscara de la luz.*

Obtener y preprocesar la componente tono. La siguiente operación que se realiza consiste en obtener la componente de color llamada tono en la cual cada valor representa un tipo de color diferente. Para ello hay que pasar la ventana del espacio de color por defecto que es RGB al espacio HSL. Una vez hecha la conversión se extrae el canal del tono (*hue*) que es el contiene la información sobre los colores que aparecen en la ventana.

Figura 3.102: *Obtener la componente del tono.*

Como esta componente es inestable se suelen obtener saltos de tono muy altos en regiones que comparten el mismo tono de color. Para minimizar este efecto lo que siguiente que se hace es pasar un filtro *gaussiano* sobre la componente de tal manera que la transición de tonos sea algo más uniforme.

Figura 3.103: *Preprocesar la componente del tono.*

Obtener el tono medio. Ahora sobre la componente de tono se aplica la máscara para obtener los pixeles de interés y se calcula su histograma. Dicho histograma reflejará cuales son los tonos de color que tiene la luz del semáforo. En la siguiente imagen se muestra como se realiza este proceso:

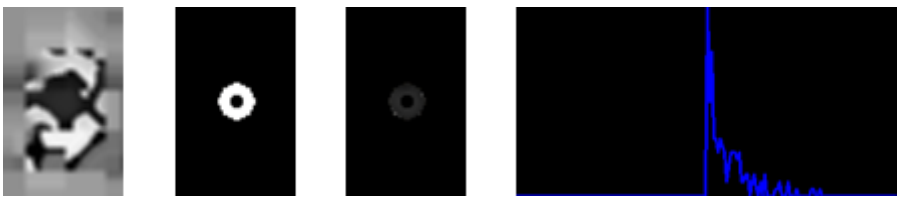


Figura 3.104: *Obtener el histograma del tono.*

Para cálculos posteriores, se van a eliminar aquellos tonos que aparecen en el histograma pero que no son importantes. Esto se hace aplicando un nivel de umbral sobre el histograma que se obtiene de multiplicar el número de pixeles máximo del histograma por un factor llamado como `hParams.histFactor`. Este factor representa la relación entre el número mínimo de pixeles aceptable y el número máximo de pixeles del histograma.

Figura 3.105: *Filtrar tonos.*

Una vez aplicado el umbral, quedarán en el histograma los tonos predominantes. Ahora pues, se va a calcular el tono medio de los valores que quedan para obtener una medida única del tono que tiene la luz del semáforo.

Figura 3.106: *Obtener el tono medio.*

En este caso, se podría haber elegido como tono de la luz aquel que tenga el mayor número de píxeles en el histograma, pero se comprobó que dicho valor era muy inestable y que daba mejores resultados la media de los tonos predominantes.

Identificar el color de la luz. Para identificar el color de la luz a través del tono medio, lo que se hace es definir unos rangos de tono entre los que se encuentran los distintos colores (rojo, ámbar y verde) que puede tener las luces del semáforo. Estos tonos se han obtenido de estudiar las muestras de semáforos que se tienen. Así pues, el color que represente el rango en el que se encuentre el tono medio será el que se asigne a la luz del semáforo (`spotLightColors[i]`). Aclarar que si el tono medio no se encuentra entre los rangos definidos se definirá el color de la luz como desconocido.

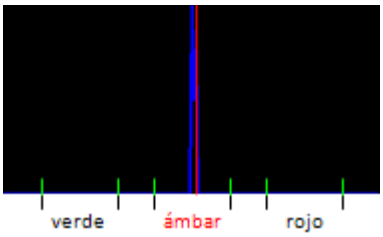


Figura 3.107: *Identificar el color de la luz.*

Llegada a este punto, la función `spotLightsRecognitionByHue()` procesaría un nuevo semáforo y así hasta completar todos los que hayan sido detectados en la imagen captada.

3.3.7. Parámetros de la Aplicación

Ahora que se conocen las funciones de la aplicación *tldr*, es posible entender el significado de los parámetros con los que dicha aplicación trabaja. Así que, lo que aquí se pretende es hacer una descripción uno a uno de todos estos parámetros para que el usuario sepa cómo utilizarlos. Aclarar que se ha decidido crear un archivo *script* llamado *script_tldr* desde el que se pueden configurar fácilmente estos parámetros, por lo que los nombres con los que aquí aparecen son con los que se encuentran en ese archivo *script*. Teniendo en cuenta esto, a continuación se presentan los parámetros organizados en categorías según las funciones que las utilizan:

Aplicación.

- **`application`**

  En este parámetro se ha de especificar la ruta del archivo ejecutable donde se implementa la aplicación *tldr*.

Fuente de entrada de imágenes.

- **`si_list_file_name`**

  Nombre del archivo de texto que contiene una lista de rutas de imágenes donde realizar la detección de semáforos.

- **`si_video_file_name`**

  Nombre del archivo de vídeo desde el que se obtienen las imágenes donde detectar semáforos.

- **`si_camera_number`**

  Número de la cámara desde la que se captarán las imágenes donde realizar la detección.

- **`si_frame_rate`**

  Tasa de captura de fotogramas. Representa cada cuantos fotogramas del archivo de vídeo o de la cámara se captura uno para procesarlo.

Detección de los semáforos.

- **`tld_cascade_file_name`**

  Nombre del archivo en formato *.xml* que contiene al clasificador de semáforos.

- **`tld_preprocess_shrink_factor`**

  Factor de reducción. Se multiplica por el tamaño de las imágenes para obtener el nuevo tamaño con el que redimensionarlas.

- **`tld_preprocess_crop_factor`**

  Factor que se multiplica por el alto de las imágenes para obtener la distancia con la que recortar las imágenes.

- **`tld_preprocess_blur_kernel_size`**

  Tamaño del operador con el que se filtran las imágenes.

- **`tld_detection_scale_factor`**

  Factor de escala con el que las imágenes van siendo redimensionadas dentro del detector en cada paso de detección.

- **`tld_detection_min_width`**

  Mínimo ancho de las ventanas con las que el detector busca a los semáforos en las imágenes.

- **`tld_detection_min_height`**

  Alto mínimo de las ventanas con las que el detector busca a los semáforos en las imágenes.

- **`tld_detection_max_width`**

  Ancho mínimo de las ventanas con las que el detector busca a los semáforos en las imágenes.

- **`tld_detection_max_height`**

  Ancho máximo de las ventanas con las que el detector busca a los semáforos en las imágenes.

- **`tld_min_bounding_boxes_group`**

  Número mínimo de vecinos que tiene que tener una marca de detección para que no sea rechazada.

- **`tld_min_bounding_boxes_overlap`**

  Porcentaje de solapamiento mínimo que debe existir entre dos marcas de detección para el que se rechaza aquella de menor puntuación.

- **`tld_max_bounding_boxes_number`**

  Número máximo de marcas de detección permitidas por imagen. Si el número de marcas es mayor que `max_bounding_boxes_number` se rechazan aquellas de menor puntuación.

Detección de la luz de los semáforos.

- **`sld_preprocess_blur_kernel_size`**

  Tamaño del operador con el que se filtran las ventanas de los semáforos detectados.

- **`sld_detection_threshold_step`**

  Paso con el que se incrementa la operación de umbralizado progresivo en el detector.

- **`sld_detection_min_threshold`**

  Valor de umbral desde el que comienza el umbralizado progresivo.

- **`sld_detection_max_threshold`**

  Valor de umbral en el que acaba el umbralizado progresivo.

- **`sld_detection_min_diameter`**

  Diámetro mínimo con el que el detector busca la luz del semáforo.

- **`sld_detection_max_diameter`**

  Diámetro máximo con el que el detector busca la luz del semáforo.

- **`sld_detection_min_circularity`**

  Circularidad mínima con la que el detector busca la luz del semáforo.

- **`sld_detection_max_circularity`**

  Circularidad máxima con la que el detector busca la luz del semáforo.

- **`sld_detection_min_inertia_ratio`**

  Relación de inercia mínima con la que el detector busca la luz del semáforo.

- **`sld_detection_max_inertia_ratio`**

  Relación de inercia máxima con la que el detector busca la luz del semáforo.

- **`sld_detection_min_convexity`**

  Convexidad mínima con la que el detector busca la luz del semáforo.

- **`sld_detection_max_convexity`**

  Convexidad máxima con la que el detector busca la luz del semáforo.

- **`sld_min_bounding_blobs_group`**

  Número mínimo de vecinos que tiene que tener una marca de detección para que no sea rechazada.

- **`sld_min_bounding_blobs_overlap`**

  Porcentaje de solapamiento mínimo que debe existir entre dos marcas de detección para el que se rechaza aquella de menor puntuación.

Reconocimiento de la luz de los semáforos.

- **`slr_recognition_type`**

  Tipo de reconocimiento de la luz de los semáforos. Las opciones que admite son por posición (`pos`), por *matching templates* (`match`), por tono del color (`hue`), por posición y tono (`pos-hue`) y por *matching* y tono (`match-hue`).

Reconocimiento de la luz de los semáforos por posición.

- **`slrp_blur_kernel_size`**

  Tamaño del operador con el que se filtran las ventanas de los semáforos detectados.

- **`slrp_sobel_kernel_size`**

  Tamaño del operador de Sobel con el que se obtienen los bordes horizontales presentes en las ventanas de los semáforos.

- **`slrp_sobel_edges_threshold`**

  Umbral que se aplica a los bordes encontrados por el operador de Sobel para obtener aquellos más importantes.

- **slrp_bound_lines_of_lights_factor**

  Factor que se multiplica por el diámetro de las luces detectadas para delimitar los bordes de dichas luces y que no se confundan con los bordes de los semáforos.

- **slrp_edges_width_factor**

  Factor que se multiplica por el diámetro de las luces detectadas para obtener el ancho mínimo que han de tener los bordes para que pertenezcan a los semáforos.

- **slrp_distance_between_lights_factor**

  Factor que indica la relación que tiene la distancia entre dos lámparas de un semáforo y el diámetro de esas lámparas.

Reconocimiento de la luz de los semáforos por *matching templates*.

- **slrm_red_template_file_name**

  Nombre del archivo imagen que contiene la plantilla correspondiente a la luz roja.

- **slrm_amber_template_file_name**

  Nombre del archivo imagen que contiene la plantilla correspondiente a la luz ámbar.

- **slrm_green_template_file_name**

  Nombre del archivo imagen que contiene la plantilla correspondiente a la luz verde.

- **slrm_red_light_box_x_location**

  Coordenada x de la esquina superior izquierda de la región que representa la luz del semáforo en la plantilla de luz roja.

- **slrm_red_light_box_y_location**

  Coordenada y de la esquina superior izquierda de la región que representa la luz del semáforo en la plantilla de luz roja.

- **slrm_red_light_box_width_size**

  Ancho de la región que representa la luz del semáforo en la plantilla de luz roja.

- **slrm_red_light_box_height_size**

  Alto de la región que representa la luz del semáforo en la plantilla de luz roja.

- **slrm_amber_light_box_x_location**

  Coordenada x de la esquina superior izquierda de la región que representa la luz del semáforo en la plantilla de luz ámbar.

- **slrm_amber_light_box_y_location**

  Coordenada y de la esquina superior izquierda de la región que representa la luz del semáforo en la plantilla de luz ámbar.

- **slrm_amber_light_box_width_size**

  Ancho de la región que representa la luz del semáforo en la plantilla de luz ámbar.

- **slrm_amber_light_box_height_size**

  Alto de la región que representa la luz del semáforo en la plantilla de luz ámbar.

- **`slrm_green_light_box_x_location`**

  Coordenada x de la esquina superior izquierda de la región que representa la luz del semáforo en la plantilla de luz verde.

- **`slrm_green_light_box_y_location`**

  Coordenada y de la esquina superior izquierda de la región que representa la luz del semáforo en la plantilla de luz verde.

- **`slrm_green_light_box_width_size`**

  Ancho de la región que representa la luz del semáforo en la plantilla de luz verde.

- **`slrm_green_light_box_height_size`**

  Alto de la región que representa la luz del semáforo en la plantilla de luz verde.

- **`slrm_blur_kernel_size`**

  Tamaño del operador con el que se filtran las regiones de los semáforos donde se aplica el *matching*.

Reconocimiento de la luz de los semáforos por tono color

- **`slrh_blur_kernel_size`**

  Tamaño del operador con el que se filtra el canal de tono de las ventanas de los semáforos detectados.

- **`slrh_internal_halo_factor`**

  Factor que se multiplica por el diámetro de las luces de los semáforos detectados para obtener el diámetro interno de la máscara que se aplica sobre las ventanas de detección.

- **`slrh_external_halo_factor`**

  Factor que se multiplica por el diámetro de las luces de los semáforos detectados para obtener el diámetro externo de la máscara que se aplica sobre las ventanas de detección.

- **`slrh_filter_histogram_factor`**

  En el histograma del tono de las luces detectadas, este factor indica, en porcentaje sobre el máximo número de pixeles, el número mínimo de pixeles que ha de tener un tono para que se utilice en el cálculo del tono medio.

- **`slrh_max_red_hue`**

  Tono máximo del rango de tonos con el que se identifica a la luz roja.

- **`slrh_min_red_hue`**

  Tono mínimo del rango de tonos con el que se identifica a la luz roja.

- **`slrh_max_amber_hue`**

  Tono máximo del rango de tonos con el que se identifica a la luz ámbar.

- **`slrh_min_amber_hue`**

  Tono mínimo del rango de tonos con el que se identifica a la luz ámbar.

- **`slrh_max_green_hue`**

  Tono máximo del rango de tonos con el que se identifica a la luz verde.

- **`slrh_min_green_hue`**

  Tono mínimo del rango de tonos con el que se identifica a la luz verde.

Visualización de la detección y el reconocimiento de los semáforos.

- **`drd_red_image_file_name`**

  Nombre de archivo de la imagen que se utiliza para representar en la ventana de visualización a un semáforo que se ha identificado con luz roja.

- **`drd_amber_up_image_file_name`**

  Nombre de archivo de la imagen que se utiliza para representar en la ventana de visualización a un semáforo que se ha identificado con luz ámbar en la posición central.

- **`drd_green_image_file_name`**

  Nombre de archivo de la imagen que se utiliza para representar en la ventana de visualización a un semáforo que se ha identificado con luz verde.

- **`drd_amber_down_image_file_name`**

  Nombre de archivo de la imagen que se utiliza para representar en la ventana de visualización a un semáforo que se ha identificado con luz ámbar en la posición inferior.

- **`drd_amber_or_green_image_file_name`**

  Nombre de archivo de la imagen que se utiliza para representar en la ventana de visualización a un semáforo que se ha identificado con luz en la posición inferior.

- **`drd_amber_up_or_down_image_file_name`**

  Nombre de archivo de la imagen que se utiliza para representar en la ventana de visualización a un semáforo que se ha identificado con luz ámbar en cualquier posición.

- **`drd_unknown_image_file_name`**

  Nombre de archivo de la imagen que se utiliza para representar en la ventana de visualización a un semáforo que se ha identificado con luz desconocida.

- **`drd_detection_draw_color`**

  Color con el que se dibujan las marcas de detección sobre la ventana de visualización.

- **`drd_detection_draw_thickness`**

  Espesor de la línea con la que se dibujan las marcas de detección sobre la ventana de visualización.

- **`drd_display_scale_factor`**

  Factor de escala con el que se muestran las imágenes captadas en la ventana de visualización.

- **`help`**

  Parámetro que indica a la aplicación *tldr* que muestre por pantalla información acerca del uso de sus parámetros. Su valor puede ser verdadero (`true`) o falso (`false`).

3.3.8. Ejecución de la Aplicación

Para ejecutar la aplicación tldr lo que habría que hacer es abrir el archivo *script* que trabaja con la aplicación y que se llama *script_tldr*, editar la ruta donde se encuentra la aplicación *tldr* y configurar sus parámetros. Una vez guardados los cambios, si se dan privilegios de ejecución al archivo *script* y se hace doble clic sobre él, se abrirá una ventana del terminal y comenzará a ejecutarse la aplicación *tldr*. En dicho terminal aparecerá la siguiente información:

```
+------------------------------------------------------------------------------+
|                  TRAFFIC LIGHTS DETECTION AND RECOGNITION                    |
|                          By Victor Alonso Mendieta                           |
|                                October 2013                                  |
|                                                                              |
|                  Traffic lights detection and recognition                    |
|              using cascade classifier with HAAR-LBP features.               |
|                     Compiled with OpenCV version 2.4.6                       |
+------------------------------------------------------------------------------+
Program parameters...

  Source input parameters:

    List file: Files/list.txt
    Video file: null
    Camera number: 0
    Frame rate: 5

  Traffic lights detection parameters:

    Cascade file: Files/cascade.xml
    Shrink factor: 0.5
    Crop factor: 0.5
    Blur size: 3
    Scale factor: 1.1
    Min width: 0
    Min height: 0
    Max width: 0
    Max height: 0
    Min group: 1
    Min overlap: 0
    Max number: 3

  Spot lights detection parameters:

    Blur size: 3
    Threshold step: 5
    Min threshold: 0
    Max threshold: 254
    Min diameter: 10
    Max diameter: 40
    Min circularity: 0.8
    Max circularity: 1
    Min inertia ratio: 0.8
    Max inertia ratio: 1.2
    Min convexity: 0.8
    Max convexity: 1.2
    Min group: 1
    Min overlap: 0

  Spot lights recognition parameters:

    Recognition type: pos-hue

  Spot lights recognition by position parameters:

    Blur size: 3
    Sobel size: 3
    Edges trhesh: 100
```

```
    Bound factor: 0.25
    Width factor: 0.8
    Distance factor: 1.5

  Spot lights recognition by matching parameters:

    Red template file: Files/Matching_Templates/red.jpg
    Amber template file: Files/Matching_Templates/amber.jpg
    Green template file: Files/Matching_Templates/green.jpg
    Red light box x: 0
    Red light box y: 0
    Red light box width: 35
    Red light box height: 35
    Amber light box x: 0
    Amber light box y: 45
    Amber light box width: 35
    Amber light box height: 35
    Green light box x: 0
    Green light box y: 90
    Green light box width: 35
    Green light box height: 35
    Blur size: 3

  Spot lights recognition by hue parameters:

    Blur size: 3
    Internal halo factor: 0
    External halo factor: 1.25
    Histogram factor: 0.5
    Max red hue: 150
    Min red hue: 125
    Max amber hue: 120
    Min amber hue: 90
    Max green hue: 40
    Min green hue: 30

  Detection and recognition display:

    Red image file: Files/Recognition_Images/red.jpg
    Amber image file: Files/Recognition_Images/amberup.jpg
    Green image file: Files/Recognition_Images/green.jpg
    Amber down image file: Files/Recognition_Images/amberdown.jpg
    Amber or green image file: Files/Recognition_Images/amberorgreen.jpg
    Amber up or down image file: Files/Recognition_Images/amberupordown.jpg
    Unknown image file: Files/Recognition_Images/unknown.jpg
    Draw color: blue
    Draw thickness: 4
    Scale factor: 0.5

+--------------------------------------------------------------------------+
Checking program parameters...

  Done

+--------------------------------------------------------------------------+
Loading program files and devices...

  Done

+--------------------------------------------------------------------------+
Start detection and recognition (Press ESC to exit)...
```

Figura 3.108: *Información mostrada en el terminal al ejecutar script_tldr.*

Entre la información que aparece en el terminal se encuentra el título de la aplicación, la lista de parámetros junto con los valores introducidos, el resultado de comprobar que los parámetros son correctos, el resultado de cargar los archivos requeridos y por último un mensaje con el que se comienza a ejecutar la función principal de la aplicación. Al imprimirse ese mensaje, se abre una ventana donde se irá mostrando los resultados de la detección y el reconocimiento de semáforos mediante la aplicación. En dicha ventana se pueden distinguir dos subventanas:

- **Ventana de detección.** En la ventana de detección se irán mostrando las imágenes captadas (de la lista, del vídeo o de la cámara) junto con las marcas de detección dibujadas tanto de los semáforos (rectángulos) como de sus luces (círculos) que indican la ubicación de estos.

- **Ventana de reconocimiento.** En la parte inferior hay reservada una zona en la que se irán mostrando unas imágenes que representan la señalización de los semáforos marcados de en la ventana de detección según aparecen de derecha a izquierda.

En la siguiente imagen se muestra a la ventana de detección y reconocimiento de semáforos en funcionamiento:

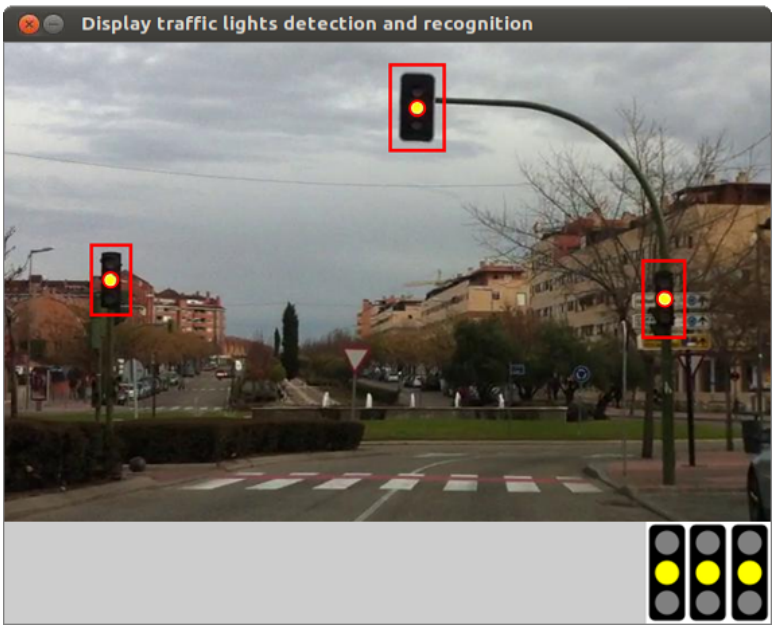


Figura 3.109: *Ventana de detección y reconocimiento de semáforos.*

Anteriormente no se ha comentado pero las señalizaciones que se muestran en la ventana de reconocimiento van a depender del tipo de reconocimiento que se haya elegido (posición, *matching*, tono, posición-tono y *matching*-tono) puesto que en algunos de estos tipos solo se puede reconocer o bien la posición de la luz pero no el color, o bien el color pero no la posición o ambas. Así pues, se han tenido que crear nuevos tipos de señalizaciones que representen estos casos. A continuación se listan los tipos de reconocimiento junto con las respectivas señalizaciones que pueden proporcionar:

- **Reconocimiento por posición.** Como en este tipo de reconocimiento solo se puede conocer la posición de la luz respecto del semáforo y los semáforos a detectar son de tres posiciones (abajo, centro, arriba) existen tres tipos de señalizaciones para cubrir esas posibles posiciones (ámbar o verde, ámbar y rojo) y una más para aquel caso en el que no se haya podido identificar la posición (desconocido). La razón de que la posición inferior se la denomine como ámbar o verde es debido a que en los semáforos estudiados se pueden dar ambas señalizaciones y como aquí no se detecta el color pues pueden darse las dos posibilidades.

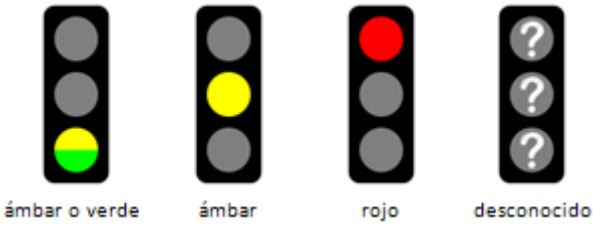


Figura 3.110: *Señalizaciones para el reconocimiento por posición.*

- **Reconocimiento por *matching*.** Aquí también se detecta solo la posición de la luz de los semáforos por lo que el tipo de señalización será el mismo que en el tipo de reconocimiento por posición. Se tendrán entonces las tres señalizaciones (ámbar o verde, ámbar y rojo) para cada una de las posiciones (abajo, centro, arriba) y otra más para indicar que no se ha podido identificar ninguna (desconocido).

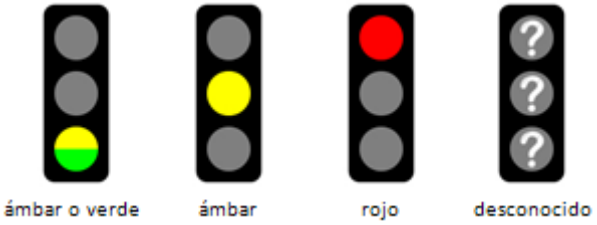


Figura 3.111: *Señalizaciones para el reconocimiento por matching.*

- **Reconocimiento por tono.** En el caso del reconocimiento por tono, tan solo se puede identificar el color de la luz del semáforo pero no su posición, por lo que las opciones se limitan tan solo a los posibles colores de dicha luz (verde, ámbar, rojo). Como el color ámbar se puede dar en dos posiciones (centro o abajo), de ahí que se denomine a esa señalización como ámbar centro o abajo. Además existe un tipo de señalización para designar a aquella luz a la que no se le ha podido asignar ningún color.

Figura 3.112: *Señalizaciones para el reconocimiento por tono.*

- **Reconocimiento por posición y tono.** Con este tipo de reconocimiento es posible identificar tanto la posición de la luz como su color, por lo que ahora si se dará la señalización igual a como la da del semáforo. Así pues, se tienen cuatro tipos de señalizaciones que son verde, ámbar abajo, ámbar centro, rojo y una más que es desconocido para indicar que no se ha podido reconocer ninguna.

Figura 3.113: *Señalizaciones para el reconocimiento por posición y tono.*

- **Reconocimiento por posición y tono.** Aquí, igual que en el caso anterior, se detecta tanto la posición como el color de la luz por lo que las señalizaciones son las propias del semáforo. Estas señalizaciones son verde, ámbar abajo, ámbar centro, rojo y desconocido.

Figura 3.114: *Señalizaciones para el reconocimiento por matching y tono.*

Con toda la información proporcionada hasta aquí, se daría por concluida la descripción de la aplicación *tldr* con la que se pretende que cualquier usuario con unos conocimientos básicos pueda aprender a manejarla.

Capítulo 4

Conclusiones y Trabajos Futuros

4.1. Conclusiones

En este apartado se tratará de hacer un repaso de las conclusiones más importantes obtenidas en el desarrollo del proyecto.

- De la recopilación de imágenes, se puede decir que la idea de adquirir las muestras de semáforos a partir de grabaciones del tráfico tomadas desde una cámara acoplada a un vehículo ha resultado ser positiva. Esto ha permitido obtener un gran número de muestras con las que realizar el entrenamiento y así obtener un clasificador con un muy buen rendimiento. El único inconveniente en este proceso es que se ha utilizado una cámara de baja calidad como es la de un móvil, con la que las luces de los semáforos producían efectos de halo y deslumbramiento sobre las imágenes captadas. Más tarde se vio que estos efectos dificultaban el estudio de una técnica que permitiese reconocer bien el color de las luces. Este problema podría haberse evitado si se hubiese utilizado una cámara profesional. En cualquier caso, la calidad de las imágenes ha sido lo suficientemente buena para entrenar al clasificador y poder desarrollar la aplicación.

- Una de las partes que se considera clave del proyecto es el diseño de la aplicación *Marcador de Objetos*. El interfaz de usuario y las funciones que implementa esta aplicación han permitido realizar la laboriosa tarea de etiquetar las muestras de semáforos de forma cómoda y rápida. Si no fuera por esta aplicación hubiese sido imposible el haber podido obtener tantas muestras de semáforos con las que entrenar al clasificador, y en definitiva esta es la razón de que el clasificador haya resultado tener ese tan buen rendimiento.

- En el entrenamiento, comentar que el modelo de directorio que se ha planteado, donde se organizan los archivos y aplicaciones involucrados en el mismo, ha ayudado a facilitar todas la tares relacionadas con este proceso. Como ya se ha comentado en otras ocasiones, la configuración y ejecución las aplicaciones ahora se simplifica a la simple tarea de editar y a hacer doble clic sobre unos archivos *script,* evitando así lo tedioso que es trabajar sobre la línea de comandos. Las ventajas de esta forma de trabajar se aprecian cuando se tienen que realizar varios entrenamientos como es en el caso de este proyecto. Además, estos archivos han permitido añadir otras funcionalidades como medir el tiempo que tarda en ejecutarse un entrenamiento o generar informes de resultados.

- Por otro lado, las aplicaciones que se han programado para preprocesar las imágenes de muestra y para evaluar al clasificador, han añadido esas operaciones que se creen que le faltaban al proceso de entrenamiento mediante las aplicaciones proporcionadas por *OpenCV*. Con la aplicación de preprocesar imágenes se ha conseguido comprobar el efecto que tiene el mejorar la calidad de las muestras sobre el comportamiento del clasificador entrenado con ellas. La aplicación de evaluar al clasificador ha permitido analizar el rendimiento de los clasificadores entrenados con lo que se han podido ajustar los parámetros de entrenamiento para obtener el mejor clasificador posible.

- Analizando la aplicación de detección y reconocimiento de semáforos, la parte que mejores resultados ha demostrado obtener ha sido la de la detección del semáforo. La razón se debe a que todo el trabajo se ha centrado en entrenar a un clasificador con buen rendimiento porque se sabía que este era la clave de todo lo que se fuese a desarrollar en torno a él después. En cuanto al reconocimiento de la señalización de semáforos, quizá sea la parte mejorable de la aplicación. Pero no hay que olvidar que se han utilizado técnicas simples que han surgido sin más de analizar las imágenes del semáforo. Probablemente aplicando algún tipo de algoritmo más complejo y específico para este tipo de aplicaciones se obtengan mejores resultados. Aún así, las técnicas utilizadas han logrado dar buenos resultados.

- Por falta de tiempo, no se han podido realizar pruebas de rendimiento que realmente demuestren la calidad de la aplicación de detección y reconocimiento de semáforos, aunque a simple vista se ha podido comprobar que es una aplicación bastante eficaz y fiable. Con esto lo que se pretende indicar es que para las imágenes que se le han pasado, detecta y reconoce en gran mayoría los semáforos que en ellas aparecen sin apenas cometer errores. En cuanto a si la aplicación puede funcionar en tiempo real, antes habría que realizar un estudio de tiempos para ver cuánto se tarda en procesar una imagen y con ello comprobar si realmente lo es. Decir que a simple vista las pruebas realizadas sobre archivos de vídeo demuestran que sí, por lo que no debería de haber problemas con imágenes tomadas desde una cámara. Además, es un parámetro que va a depender de la velocidad de procesamiento del equipo que vaya a instalar la aplicación.

En definitiva, la conclusión más importante a la que se llega es que la aplicación diseñada cumple con las expectativas que se habían marcado. Esto significa que la aplicación es capaz de detectar semáforos y de reconocer su señalización en tiempo real con un elevado grado de eficacia y seguridad.

4.2. Trabajos Futuros

Tras cumplir con los objetivos marcados para este proyecto, se podrían plantear una serie de trabajos futuros que tomen como partida el trabajo que aquí se presenta y que tengan como finalidad el mejorarlo o el darle aplicaciones.

- Quizá por ser la parte que no tan buenos resultados ha dado, uno de los trabajos futuros a realizar sería el mejorar los métodos utilizados para reconocer la señalización de los semáforos. Para ello, se podrían utilizar algoritmos más complejos y profesionales con los que identificar la posición de la luz de los semáforos y su color.

- Una de las partes que ha quedado pendiente en este proyecto es medir el rendimiento de la aplicación de detección y reconocimiento de semáforos. Así pues, sería necesario diseñar una aplicación que integre a la primera, en la que se le vayan pasando imágenes con semáforos, a los que ya se tiene identificada su señalización, y se fuese indicando si el reconocimiento que se realiza es correcto o no. Así pues, si en un futuro se continuase desarrollando la aplicación este sería una de las tareas por las que se debería comenzar.

- También habría que probar la aplicación de detección y reconocimiento de semáforos directamente desde un vehículo circulando. Para ello, se requeriría de un vehículo en el que acoplar un equipo que instalase dicha aplicación y una cámara conectada a este equipo desde la que captar las imágenes del tráfico. De este trabajo de campo se podrían extraer conclusiones acerca de la validez de la aplicación construida y de las posibles mejoras que se podrían implementar. En definitiva, lo que se pretendería con todo este trabajo es llegar a completar la aplicación para que en futuro se pudiese implantar de forma comercial en los vehículos.

- Otra de los posibles trabajos futuros que se pueden pensar para este proyecto, es crear una aplicación móvil. Esta aplicación estaría diseñada para que con cualquier móvil acoplado mediante un soporte a la luna de un vehículo se pudiese indicar al conductor la ubicación y señalización de los semáforos. Para desarrollar la aplicación móvil, tan solo habría que diseñar un interfaz de usuario y adaptar el código fuente para que funcionase con él. En principio la aplicación se colgaría en las principales tiendas de aplicaciones móviles para su descarga de forma gratuita y según su éxito se podría plantear incluso el ponerle un precio a la descarga.

Capítulo 5

Código Fuente

5.1. *preprocessimages.cpp*

```
#include <iostream>
#include <fstream>

#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"

using namespace std;
using namespace cv;

int main(int argc, char **argv)
{
   //Define parameters variables

   string listFileName;
   string preprocImagesDir;
   bool resize = false;
   double resizeFactor = 0.5;
   bool equalize = false;
   bool filter = false;
   int diameter = 5;
   int sigmaColor = 10;
   int sigmaSpace = 10;
   bool show = false;
   double showFactor=1;
   bool help = false;

   //Check number of parameters

   if( argc == 1 )
   {
      cerr << "No parameters." << endl;
      return 1;
   }

   //Get parameters introduced

   for( int i = 1; i < argc; i++ )
   {
      if( !strcmp( argv[i], "-list" ) )
      {
         listFileName = argv[++i];
      }
      else if( !strcmp( argv[i], "-directory" ) )
      {
         preprocImagesDir = argv[++i];
      }
      else if( !strcmp( argv[i], "-resize" ) )
      {
         resize = true;
      }
      else if( !strcmp( argv[i], "-resizeFactor" ) )
      {
         resizeFactor = atof( argv[++i] );
      }
      else if( !strcmp( argv[i], "-equalize" ) )
      {
         equalize = true;
      }
      else if( !strcmp( argv[i], "-filter" ) )
      {
         filter = true;
      }
      else if( !strcmp( argv[i], "-diameter" ) )
      {
         diameter = atoi( argv[++i] );
      }
      else if( !strcmp( argv[i], "-sigmaColor" ) )
      {
         sigmaColor = atoi( argv[++i] );
      }
```

```
        else if( !strcmp( argv[i], "-sigmaSpace" ) )
        {
            sigmaSpace = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-show" ) )
        {
            show = true;
        }
        else if( !strcmp( argv[i], "-showFactor" ) )
        {
            showFactor = atof( argv[++i] );
        }
        else if( !strcmp( argv[i], "-help" ) )
        {
            help = true;
        }
        else
        {
            cerr <<  "Wrong parameter." << endl;
            return 1;
        }
    }

    //Print help usage

    if( help )
    {
        cout << "Usage:" << endl
             << endl
             << "  [-list <list_file_name>]" << endl
             << "  [-directory <preprocessed_images_directory>]" << endl
             << "  [-resize]" << endl
             << "  [-resizeFactor <resize_factor>]" << endl
             << "  [-equalize]" << endl
             << "  [-filter]" << endl
             << "  [-diameter <filter_diameter>]" << endl
             << "  [-sigmaColor <filter_sigma_color>]" << endl
             << "  [-sigmaSpace <filter_sigma_space>]" << endl
             << "  [-show]" << endl
             << "  [-showFactor <show_factor>]" << endl
             << "  [-help]" << endl
             << endl;

        return 0;
    }

    //Print parameters introduced

    cout << "Parameters:" << endl
         << endl
         << "  List file name: " << (( listFileName.empty() ) ? "null" : listFileName)
         << endl
         << "  Preprocessed images directory: "
         << (( preprocImagesDir.empty() ) ? "null" :  preprocImagesDir) << endl
         << "  Resize: " << (( resize ) ? "true" : "false") << endl
         << "  Resize factor : " << resizeFactor << endl
         << "  Equalize: " << (( equalize ) ? "true" : "false") << endl
         << "  Filter: " << (( filter ) ? "true" : "false") << endl
         << "  Filter diameter: " << diameter << endl
         << "  Filter sigma color: " << sigmaColor << endl
         << "  Filter sigma space: " << sigmaSpace << endl
         << "  Show: " << (( show ) ? "true" : "false") << endl
         << "  Show factor: " << showFactor << endl
         << endl;

    //Check parameters values

    if( listFileName.empty() )
    {
        cout << "Unable to access the list file." << endl;
        return 1;
    }
```

```
    if( preprocImagesDir.empty() )
    {
        cout << "Unable to access the preprocessed images directory." << endl;
        return 1;
    }

    if( !resize && !equalize && !filter )
    {
        cout << "Nothing to do." << endl;
        return 0;
    }
    else if( show )
    {
        cout << "Preprocess images (press ESC to close the display window)..." << endl
             << endl;
    }
    else if( !show )
    {
        cout << "Preprocess images..." << endl << endl;
    }

    //Open list file

    ifstream listFile;
    listFile.open( listFileName.c_str() );
    if( !listFile.is_open() )
    {
        cerr << "Unable to open list file: " << listFileName << endl;
        return 1;
    }

    //Start preprocess images

    int numImages=0;

    while( !listFile.eof() )
    {
        //Read list line

        string listLine;
        getline( listFile, listLine );
        if( listLine.empty() )
         continue;

        //Read image file name

        size_t foundIndex = listLine.find_first_of( '\t' );
        string imageName = listLine.substr( 0, foundIndex );

        //Read image

        Mat sourceImage = imread(imageName);
        if( sourceImage.empty() )
        {
            cerr << "  Unable to read image: " << imageName << endl;
            return 1;
        }
        Mat temporalImage = sourceImage;

        //Convert image to greyscale

        if( temporalImage.channels() > 1 )
        {
            Mat grayImage;
            cvtColor( temporalImage, grayImage, CV_BGR2GRAY );
            temporalImage = grayImage;
        }

        //Resize image

        if( resize )
        {
            Mat resizedImage;
```

```
            Size resizedImageSize( cvRound( temporalImage.cols*resizeFactor ),
                                   cvRound( temporalImage.rows*resizeFactor ) );
            cv::resize(temporalImage, resizedImage, resizedImageSize);
            temporalImage = resizedImage;
        }
        //Equalize image

        if( equalize )
        {
            Mat equalizedImage;
            equalizeHist(temporalImage, equalizedImage);
            temporalImage = equalizedImage;
        }

        //Filter image

        if( filter )
        {
            Mat filteredImage;
            bilateralFilter(temporalImage, filteredImage, diameter, sigmaColor, sigmaSpace);
            temporalImage = filteredImage;
        }

        //Show image

        if( show )
        {
            Mat shownImage;
            Size shownImageSize( cvRound( temporalImage.cols*showFactor ),
                                 cvRound( temporalImage.rows*showFactor ) );
            cv::resize( temporalImage, shownImage, shownImageSize );

            namedWindow("Preprocessed image");
            imshow("Preprocessed image", shownImage);

            if( waitKey( 0 ) == 27 )
            {
                show = false;
                destroyWindow("Preprocessed image");
            }
        }

        //Ignore labeled marks

        foundIndex = imageName.find_last_of("/\\");
        string fileName = imageName.substr(foundIndex);

        //Write image

        string newImageName = preprocImagesDir + fileName;
        bool writeResult = imwrite(newImageName, temporalImage);
        if( !writeResult )
        {
            cerr << "  Unable to write image: " << newImageName <<endl;
            return 1;
        }

        numImages++;

    }

    //End preprocess images

    cout << "  Done. Preprocessed " << numImages << " images." << endl;

    listFile.close();

    return 0;
}
```

5.2. *testcascade.cpp*

```
#include <iostream>
#include <iomanip>
#include <fstream>
#include <vector>
#include <algorithm>
#include <ctime>

#include "opencv2/core/core.hpp"
#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"

using namespace std;
using namespace cv;

int main( int argc, const char** argv )
{
   //Define parameters variables

   string cascadeFileName = "null";
   string listFileName = "null";
   string dataFileName = "null";
   double scaleFactor = 1.1;
   Size minObjectSize = Size();
   Size maxObjectSize = Size();
   int groupThreshold = 1;
   double minNmsOverlap = 0.5;
   double minMatchOverlap = 0.5;
   string curveType = "all";
   bool showTest = false;
   bool help = false;

   //Check number of parameters

   if( argc == 1 )
   {
      cerr << "No parameters. Use the parameter '-help' for usage." << endl;
      return 1;
   }

   //Get parameters introduced

   for( int i = 1; i < argc; i++ )
   {
      if( !strcmp( argv[i], "-cascade" ) )
      {
         cascadeFileName = argv[++i];
      }
      else if( !strcmp( argv[i], "-list" ) )
      {
         listFileName = argv[++i];
      }
      else if( !strcmp( argv[i], "-data" ) )
      {
         dataFileName = argv[++i];
      }
      else if( !strcmp( argv[i], "-scale" ) )
      {
         scaleFactor = atof( argv[++i] );
      }
      else if( !strcmp( argv[i], "-minsize" ) )
      {
         minObjectSize.width = atoi( argv[++i] );
         minObjectSize.height = atoi( argv[++i] );
      }
      else if( !strcmp( argv[i], "-maxsize" ) )
      {
         maxObjectSize.width = atoi( argv[++i] );
         maxObjectSize.height = atoi( argv[++i] );
      }
```

```
        else if( !strcmp( argv[i], "-group" ) )
        {
            groupThreshold = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-noverlap" ) )
        {
            minNmsOverlap = atof( argv[++i] );
        }
        else if( !strcmp( argv[i], "-moverlap" ) )
        {
            minMatchOverlap = atof( argv[++i] );
        }
        else if( !strcmp( argv[i], "-curve" ) )
        {
            curveType = argv[++i];
        }
        else if( !strcmp( argv[i], "-show" ) )
        {
            showTest = true;
        }
        else if( !strcmp( argv[i], "-help" ) )
        {
            help = true;
        }
        else
        {
            cerr <<  "Wrong parameter: " << argv[i] << endl;
            return 1;
        }
    }

    //Print help usage

    if( help )
    {
        cout << "Usage:" << endl
             << endl
             << "  [-cascade <cascade_file_name>]" << endl
             << "  [-list <list_file_name>]" << endl
             << "  [-data <data_file_name>]" << endl
             << "  [-scale <scale_factor>]" << endl
             << "  [-minsize <min_object_width> <min_object_height>]" << endl
             << "  [-maxsize <max_object_width> <max_object_height>]" << endl
             << "  [-group <group_threshold>]" << endl
             << "  [-noverlap <min_nms_overlap>]" << endl
             << "  [-moverlap <min_match_overlap>]" << endl
             << "  [-curve <{roc|det|pr|all}>]" << endl
             << "  [-show]" << endl
             << "  [-help]" << endl
             << endl;

        return 0;
    }

    //Print parameters introduced

    cout << "Parameters:" << endl
         << endl
         << "  Cascade file name: " << cascadeFileName << endl
         << "  List file name: " << listFileName << endl
         << "  Data file Name: " << dataFileName << endl
         << "  Scale factor: " << scaleFactor << endl
         << "  Min objects size: " << minObjectSize.width << "x" << minObjectSize.height
         << endl
         << "  Max objects size: " << maxObjectSize.width << "x" << maxObjectSize.height
         << endl
         << "  Group threshold: " << groupThreshold << endl
         << "  Min nms overlap: " << minNmsOverlap << endl
         << "  Min match overlap: " << minMatchOverlap << endl
         << "  Curve type: " << curveType << endl
         << "  Show test: " << ((showTest)? "true" : "false") << endl
         << endl;
```

```
    //Check parameters values

    if( cascadeFileName == "null" )
    {
       cerr << "Unable to access the cascade file." << endl;
       return 1;
    }

    if( listFileName == "null" )
    {
       cerr << "Unable to access the list file." << endl;
       return 1;
    }

    if( dataFileName == "null" )
    {
       cerr << "Unable to access the data file." << endl;
       return 1;
    }

    if( groupThreshold < 1 )
    {
       cerr << "Invalid group threshold." << endl;
       return 1;
    }

    if( minNmsOverlap <= 0 || minNmsOverlap > 1 )
    {
       cerr << "Invalid min nms overlap." << endl;
       return 1;
    }

    if( minMatchOverlap <= 0 || minMatchOverlap > 1 )
    {
       cerr << "Invalid min match overlap." << endl;
       return 1;
    }

    if( curveType != "roc" && curveType != "det" && curveType != "pr" && curveType != "all" )
    {
       cerr << "Invalid curve type." << endl;
       return 1;
    }

    //Load files

    CascadeClassifier cascade;
    if( !cascade.load( cascadeFileName ) )
    {
       cerr << "Unable to load casacade classifier: " << cascadeFileName << endl;
       return 1;
     }

    ifstream listFile;
    listFile.open( listFileName.c_str() );
    if( !listFile.is_open() )
    {
       cerr << "Unable to load list file: " << listFileName << endl;
       return 1;
    }

    ofstream dataFile;
    dataFile.open( dataFileName.c_str() );
    if( !dataFile.is_open() )
    {
       cerr << "Unable to load data file: " << dataFileName << endl;
       return 1;
    }



    //Define program variables
```

```
vector<double> totalMarkScores;
vector<int> totalHits;
vector<int> totalMisses;
vector<int> totalTruePositives;
vector<int> totalFalsePositives;

int totalNumHits = 0;
int totalNumMisses = 0;
int totalNumFalsePositives = 0;
int totalNumRealPositives = 0;

//Start cascade test

cout << "Cascade test..." << endl
     << endl;

cout << "  +----------------------+--------+--------+--------+" << endl
     << "  | File Name            |     TP |     FN |     FP |" << endl
     << "  +----------------------+--------+--------+--------+" << endl;

if( showTest )
{
   namedWindow( "Show Test" );
}

 while( !listFile.eof() )
{
   //Read list line

   string listLine;
   getline( listFile, listLine );
   if( listLine.empty() )
    continue;

   //Read image name and number of objects fields

   string imagePath;
   int numObjects;
   istringstream listStream( listLine );
   listStream >> imagePath;
   listStream >> numObjects;

   //Read labeled marks

   vector<Rect> labeledMarks( numObjects, Rect() );
   for( int i = 0; i < numObjects; i++ )
   {
      listStream >> labeledMarks[i].x;
      listStream >> labeledMarks[i].y;
      listStream >> labeledMarks[i].width;
      listStream >> labeledMarks[i].height;
   }
   int labeledSize = labeledMarks.size();

   //Read image

   Mat image = imread( imagePath, CV_LOAD_IMAGE_UNCHANGED );
   if( image.empty() )
   {
      cerr << "  Unable to read image: " << imagePath << endl;
      return 1;
   }

   //Detect objects

   vector<Rect> detectedMarks;
   cascade.detectMultiScale( image, detectedMarks, scaleFactor, 0, 0, minObjectSize,
                             maxObjectSize );



   //Group detected marks
```

```
vector<int> markScores;
groupRectangles( detectedMarks, markScores, groupThreshold );
int detectedSize = detectedMarks.size();

//Sort detected marks in descending order of score

for( int i = 0; i < detectedSize - 1; i++ )
{
   for( int j = i + 1; j < detectedSize; j++ )
   {
      if( markScores[i] < markScores[j] )
      {
         swap( detectedMarks[i], detectedMarks[j] );
         swap( markScores[i], markScores[j] );
      }
   }
}

//Apply non-maximun suppression

   for( int i = 0; i < detectedSize - 1; i++ )
   {
      for( int j = i + 1; j < detectedSize; j++ )
      {
         int x1 = max( detectedMarks[i].x, detectedMarks[j].x );
         int y1 = max( detectedMarks[i].y, detectedMarks[j].y );
         int x2 = min( detectedMarks[i].x + detectedMarks[i].width - 1,
                       detectedMarks[j].x + detectedMarks[j].width - 1 );
         int y2 = min( detectedMarks[i].y + detectedMarks[i].height - 1,
                       detectedMarks[j].y + detectedMarks[j].height - 1 );

         int width = x2 - x1 + 1;
         int height = y2 - y1 + 1;

         if( width > 0 && height > 0 )
         {
            int intersectionArea = width * height;
            int unionArea =  detectedMarks[j].area() + detectedMarks[j].area() -
                             intersectionArea;
            double overlap = double( intersectionArea ) / unionArea;

            if( overlap >= minNmsOverlap )
            {
               detectedMarks.erase( detectedMarks.begin() + j );
               markScores.erase( markScores.begin() + j );
               detectedSize--;
               j--;
            }
         }
      }
   }

   //Identify hits, misses, true positives and false positives

   vector<int> hits ( labeledSize, 0 );
   vector<int> misses ( labeledSize, 0 );
   vector<int> truePositives ( detectedSize, 0 );
   vector<int> falsePositives ( detectedSize, 0 );

   for( int i = 0; i < detectedSize; i++ )
   {
      int maxIndex = 0;
      double maxOverlap = 0.0;
      for( int j = 0; j < labeledSize; j++ )
      {
         int x1 = max( detectedMarks[i].x, labeledMarks[j].x );
         int y1 = max( detectedMarks[i].y, labeledMarks[j].y );
         int x2 = min( detectedMarks[i].x + detectedMarks[i].width - 1,
                 labeledMarks[j].x + labeledMarks[j].width - 1 );
         int y2 = min( detectedMarks[i].y + detectedMarks[i].height - 1,
                 labeledMarks[j].y + labeledMarks[j].height - 1 );
```

```
                int width = x2 - x1 + 1;
                int height = y2 - y1 + 1;

                if( width > 0 && height > 0 )
                {
                    int intersectionArea = width * height;
                    int unionArea = detectedMarks[i].area() + labeledMarks[j].area() -
                                    intersectionArea;
                    double overlap = double( intersectionArea ) / unionArea;

                    if( overlap > maxOverlap )
                    {
                                                maxIndex = j;
                                                maxOverlap = overlap;
                    }
                }
            }

            if( maxOverlap >= minMatchOverlap )
            {
                if( hits[maxIndex] == 0 )
                {
                    hits[maxIndex] = 1;
                    truePositives[i] = 1;
                }
            }
        }

        for( int i = 0; i < labeledSize; i++ )
        {
                    if( hits[i] == 0 )
                            misses[i] = 1;
        }

        for( int i = 0; i < detectedSize; i++ )
        {
                    if( truePositives[i] == 0 )
                            falsePositives[i] = 1;
        }

        //Calculate number of hits, misses, false positives and real positives

        int numHits = 0;
        int numMisses = 0;
        int numFalsePositives = 0;
        int numRealPositives = 0;

        for( int i = 0; i < labeledSize; i++ )
        {
            if( hits[i] == 1 )
              numHits++;
        }

        for( int i = 0; i < labeledSize; i++ )
        {
            if( misses[i] == 1)
              numMisses++;
        }

        for( int i = 0; i < detectedSize; i++ )
        {
            if( falsePositives[i] == 1 )
              numFalsePositives++;
        }

        numRealPositives = numHits + numMisses;


        //Print results

        int foundIndex = imagePath.find_last_of( "\\/" );
        string imageName = imagePath.substr( foundIndex + 1 ) ;
```

```
            cout << "  | " << setw( 20 ) << left  << imageName.c_str()
                 <<  " | " << setw( 6 )  << right << numHits
                 <<  " | " << setw( 6 )  << right << numMisses
                 <<  " | " << setw( 6 )  << right << numFalsePositives
                 <<  " | " << endl
                 << "  +----------------------+--------+--------+--------+" << endl;

            //Show test results

            if( showTest )
            {
               for( int i = 0; i < labeledSize; i++ )
               {
                  Scalar color;

                  if( hits[i] == 1)
                  {
                     color=Scalar( 0, 255, 0 );
                  }
                  else
                  {
                     color=Scalar( 0, 0, 255 );
                  }

                     rectangle( image,
                                Point( labeledMarks[i].x, labeledMarks[i].y ),
                                Point( labeledMarks[i].x + labeledMarks[i].width,
                                       labeledMarks[i].y + labeledMarks[i].height ),
                                color, 2, CV_AA, 0 );
                  }

                  for( int i = 0; i < detectedSize; i++ )
                  {
                     Scalar color;

                     if( truePositives[i] == 1 )
                     {
                        color=Scalar( 0, 255, 255 );
                     }
                     else
                     {
                        color=Scalar( 255, 0, 0 );
                     }

                     rectangle( image,
                                Point( detectedMarks[i].x, detectedMarks[i].y ),
                                Point( detectedMarks[i].x + detectedMarks[i].width,
                                      detectedMarks[i].y + detectedMarks[i].height ),
                                color, 2, CV_AA, 0 );
                  }

                  imshow( "Show Test", image );
                  waitKey( 1 );
               }

               //Add results to total vectors

               for( int i = 0; i < detectedSize; i++)
               {
                  totalMarkScores.push_back( markScores[i] );
                  totalTruePositives.push_back( truePositives[i] );
                  totalFalsePositives.push_back( falsePositives[i] );
               }

               totalNumHits += numHits;
               totalNumMisses += numMisses;
               totalNumFalsePositives += numFalsePositives;
               totalNumRealPositives += numRealPositives;

               //Clear all used vectors

               labeledMarks.clear();
            detectedMarks.clear();
```

```
        markScores.clear();
        hits.clear();
        misses.clear();
        truePositives.clear();
        falsePositives.clear();
      }

      //Close test window

      if( showTest )
      {
        destroyWindow( "Show Test" );
      }

      //Print total results

      cout << "  | " << setw( 20 )  << left  << "Total"
           <<  " | "  << setw( 6 )  << right << totalNumHits
           <<  " | "  << setw( 6 )  << right << totalNumMisses
           <<  " | "  << setw( 6 )  << right << totalNumFalsePositives
           <<  " | " << endl
           << "  +----------------------+--------+--------+--------+" << endl
           << endl;

      //Start curves calculation

      cout << "Curves calculation..." << endl
            << endl;

      //Sort total vectors in descending order

      int totalSize = totalTruePositives.size();
      for( int i = 0; i < totalSize - 1; i++ )
      {
         for( int j = i + 1; j < totalSize; j++ )
         {
            if( totalMarkScores[i] < totalMarkScores[j] )
            {
               swap( totalMarkScores[i], totalMarkScores[j] );
               swap( totalTruePositives[i], totalTruePositives[j] );
               swap( totalFalsePositives[i], totalFalsePositives[j] );
            }
         }
      }

      //Calculate the accumulated of total vectors

      for( int i = 0; i < totalSize - 1; i++ )
      {
         totalTruePositives[i+1] += totalTruePositives[i];
      }

      for( int i = 0; i < totalSize - 1; i++ )
      {
         totalFalsePositives[i+1] += totalFalsePositives[i];
      }

      //Calculate ROC curve

      vector<double> rocFalsePositiveRate;
      vector<double> rocTruePositiveRate;
      int rocSize;
      double rocAreaUnderCurve;
      double rocEqualErrorRate;

      if( curveType == "roc" || curveType == "all" )
      {
         //False Positive Rate
         rocFalsePositiveRate.push_back( 0.0 );
         for( int i = 0; i < totalSize; i++ )
         {
            rocFalsePositiveRate.push_back( double( totalFalsePositives[i] ) /
                                            ( totalNumFalsePositives ) );
```

```
        }

        //True Positive Rate
        rocTruePositiveRate.push_back( 0.0 );
        for( int i = 0; i < totalSize; i++ )
        {

            rocTruePositiveRate.push_back( double( totalTruePositives[i] ) /
                                           ( totalNumRealPositives ) );
        }

        rocSize = rocTruePositiveRate.size();

        //Area Under Curve
        rocAreaUnderCurve = 0.0;
        for( int i = 0; i < rocSize - 1; i++ )
        {
            rocAreaUnderCurve += rocTruePositiveRate[i+1] * ( rocFalsePositiveRate[i+1] -
                                                    rocFalsePositiveRate[i] );
        }

        //Equal Error Rate
        int cutIndex = 0;
        for( int i = 0; i < rocSize; i++ )
        {
            if( ( 1 - rocFalsePositiveRate[i] ) > rocTruePositiveRate[i] )
            {
                cutIndex = i;
            }
        }
        if( rocTruePositiveRate[cutIndex] == rocTruePositiveRate[cutIndex+1] )
        {
            rocEqualErrorRate = 1 - rocTruePositiveRate[cutIndex];
        }
        else
        {
            rocEqualErrorRate = rocFalsePositiveRate[cutIndex];
        }
    }

    //Calculate DET curve

    vector<double> detFalsePositiveRate;
    vector<double> detFalseNegativeRate;
    int detSize;

    if( curveType == "det" || curveType == "all" )
    {
        //False Positive Rate
        detFalsePositiveRate.push_back( 0.0 );
        for( int i = 0; i < totalSize; i++ )
        {
            detFalsePositiveRate.push_back( double( totalFalsePositives[i] ) /
                                            ( totalNumFalsePositives ) );
        }

        //False Negative Rate
        detFalseNegativeRate.push_back( 1.0 );
        for( int i = 0; i < totalSize; i++ )
        {
            detFalseNegativeRate.push_back( 1 - ( double( totalTruePositives[i] ) /
                                                  ( totalNumRealPositives ) ) );
        }

        detSize = detFalseNegativeRate.size();
    }

    //Calculate PR curve

    vector<double> prRecall;
    vector<double> prPrecision;
    int prSize;
    double prAreaUnderCurve;
```

```
        double prAveragePrecision11;

        if( curveType == "pr" || curveType == "all" )
        {
           //Recall
           prRecall.push_back( 0.0 );
           for( int i = 0; i < totalSize; i++ )
           {
              prRecall.push_back( double( totalTruePositives[i] ) / (totalNumRealPositives) );
           }

           //Precision
           prPrecision.push_back( 1.0 );
           for( int i = 0; i < totalSize; i++ )
           {
              prPrecision.push_back( double( totalTruePositives[i] ) / ( totalTruePositives[i]
                                                                  + totalFalsePositives[i] ) );
           }

           prSize = prPrecision.size();

           //Area Under Curve
           prAreaUnderCurve = 0;
           for( int i = 0; i < prSize - 1; i++ )
           {
              prAreaUnderCurve += 0.5 * ( prPrecision[i+1] + prPrecision[i] ) *
                                  ( prRecall[i+1] - prRecall[i] );
           }

           //Interpolated 11 Points Average Precision
           vector<double> prInterpolatedRecall( 11, 0.0 );
           int interpolatedSize = prInterpolatedRecall.size();
           for( int i = 0; i < interpolatedSize; i++ )
           {
              prInterpolatedRecall[i] = 0.1 * i;
           }

           prAveragePrecision11 = 0.0;
           for( int i = 0; i < interpolatedSize; i++ )
           {
              for( int j = 0; j < prSize; j++ )
              {
                 if( prRecall[j] >= prInterpolatedRecall[i] )
                 {
                    double maxPrecision = 0.0;
                    for( int k = j; k < prSize; k++ )
                    {
                       if( prPrecision[k] > maxPrecision )
                       {
                          maxPrecision = prPrecision[k];
                       }
                    }
                    prAveragePrecision11 += maxPrecision / 11.0;
                    break;
                 }
              }
           }
        }

        //Configurate stream format

        dataFile.setf( ios::left | ios::fixed );
        dataFile.precision( 6 );

        //Write ROC curve

        if( curveType == "roc" )
        {
           //Curve headers
           dataFile << setw( 10 ) << "ROC" << "\t" << setw(10) << " " << endl;

           //Performance indicators
           dataFile << setw( 10 ) << "AUC" << "\t" << setw(10) << "EER" << endl
```

```
                << setw( 10 ) << rocAreaUnderCurve << "\t" << setw( 10 )
                << rocEqualErrorRate << endl;

        //Curve data
        dataFile << setw(10) << "FPR" << "\t" << setw(10) << "TPR" << endl;

        for( int i = 0; i < rocSize; i++ )
        {
            dataFile << setw( 10 ) << rocFalsePositiveRate[i] << "\t" << setw( 10 )
                     << rocTruePositiveRate[i] << endl;
        }
    }

    //Write DET curve

    if( curveType == "det" )
    {
        //Curve headers
        dataFile << setw( 10 ) << "DET" << "\t" << setw( 10 ) << " " << endl;

        //Curve data
        dataFile << setw( 10 ) << "FPR" << "\t" << setw( 10 ) << "FNR" << endl;

        for( int i = 0; i < detSize; i++ )
        {
            dataFile << setw( 10 ) << detFalsePositiveRate[i] << "\t" << setw( 10 )
                     << detFalseNegativeRate[i] << endl;
        }
    }

    //Write PR curve

    if( curveType == "pr" )
    {
        //Curve headers
        dataFile << setw( 10 ) << "PR" << "\t" << setw( 10 ) << " " << endl;

        //Performance indicators
        dataFile << setw( 10 ) << "AUC" << "\t" << setw( 10 ) << "AP11" << endl
                 << setw( 10 ) << prAreaUnderCurve << "\t" << setw( 10 )
                <<   prAveragePrecision11 << endl;

        //Curve data
        dataFile << setw( 10 ) << "REC" << "\t" << setw( 10 ) << "PRE" << endl;

        for( int i = 0; i < prSize; i++ )
        {
            dataFile  << setw( 10 ) << prRecall[i] << "\t" << setw( 10 ) << prPrecision[i]
                      << endl;
        }
    }

    //Write ALL curves

    if( curveType == "all" )
    {
        //Curve headers
        dataFile  << setw( 10 ) << "ROC" << "\t" << setw( 10 ) << " " << "\t"
                  << setw( 10 ) << "DET" << "\t" << setw( 10 ) << " " << "\t"
                  << setw( 10 ) << "PR"  << "\t" << setw( 10 ) << " " << endl;

        //Performance indicators
        dataFile  << setw( 10 ) << "AUC" << "\t" << setw( 10 ) << "EER"  << "\t"
                  << setw( 10 ) << ""    << "\t" << setw( 10 ) << " "    << "\t"
                  << setw( 10 ) << "AUC" << "\t" << setw( 10 ) << "AP11" << endl;

        dataFile  << setw( 10 ) << rocAreaUnderCurve << "\t" << setw( 10 )
                  << rocEqualErrorRate  << "\t"
                  << setw( 10 ) << " "                << "\t" << setw( 10 )
                  << " "                << "\t"
                  << setw( 10 ) << prAreaUnderCurve  << "\t" << setw( 10 )
                  << prAveragePrecision11 << endl;
```

```
          //Curve data
          dataFile  << setw( 10 ) << "FPR" << "\t" << setw( 10 ) << "TPR" << "\t"
                    << setw( 10 ) << "FPR" << "\t" << setw( 10 ) << "FNR" << "\t"
                    << setw( 10 ) << "REC" << "\t" << setw( 10 ) << "PRE" << endl;

          for( int i = 0; i < rocSize; i++ )
          {
             dataFile  << setw( 10 ) << rocFalsePositiveRate[i] << "\t" << setw( 10 )
                       << rocTruePositiveRate[i]  << "\t"
                       << setw( 10 ) << detFalsePositiveRate[i] << "\t" << setw( 10 )
                       << detFalseNegativeRate[i] << "\t"
                       << setw( 10 ) << prRecall[i]            << "\t" << setw( 10 )
                       << prPrecision[i]          << endl;
                }
          }

          //Close files

          listFile.close();
          dataFile.close();

          //Finish curves calculation

          cout << "  Done" << endl
               << endl;

          return 0;
}
```

5.3. *script_preprocessimages.sh*

```
#!/bin/bash

#Establecer la ruta de la aplicacion "preprocessimages"
application="../Aplicaciones/preprocessimages/build/preprocessimages"

#Definir el archivo donde se guardara el informe de "script_preprocessimages.sh"
report_file="../Informes/preprocessimages/informe.txt"

#Imprimir el titulo
echo "+-------------------------------------------------------------+"  | tee $report_file
echo "|                  SCRIPT_PREPROCESSIMAGES                    |" | tee -a $report_file
echo "+-------------------------------------------------------------+"  | tee -a $report_file

#Cargar los parametros de la aplicacion
echo "Cargando parametros..."                                          | tee -a $report_file
echo ""                                                                | tee -a $report_file
                                                                       #<default>
list_file_name="../Listas/preprocessimages/lista_positivas.txt"        #<null>
preprocessed_images_directory="../Imagenes/Coleccion"                  #<null>
resize="true"                                                          #<true>
resize_factor=0.5                                                      #<0.5>
equalize="false"                                                       #<false>
filter="false"                                                         #<false>
filter_diameter=5                                                      #<5 >
filter_sigma_color=10                                                  #<10>
filter_sigma_space=10                                                  #<10>
show="false"                                                           #<false>
show_factor=1                                                          #<1>
help="false"                                                           #<false>

parameters=""
if [ $list_file_name != "null" ]; then
        parameters="$parameters -list $list_file_name"
fi
if [ $preprocessed_images_directory != "null" ]; then
        parameters="$parameters -directory $preprocessed_images_directory"
fi
if [ $resize == "true" ]; then
        parameters="$parameters -resize"
fi
if [ $resize_factor != "-1" ]; then
        parameters="$parameters -resizeFactor $resize_factor"
fi
if [ $equalize == "true"]; then
        parameters="$parameters -equalize"
fi
if [ $filter == "true"  ]; then
        parameters="$parameters -filter"
fi
if [ $filter_sigma_color != "-1" ]; then
        parameters="$parameters -sigmaColor $filter_sigma_color"
fi
if [ $filter_sigma_space != "-1" ]; then
        parameters="$parameters -sigmaSpace $filter_sigma_space"
fi
if [ $show == "true"  ]; then
        parameters="$parameters -show"
fi
if [ $show_factor != "-1"  ]; then
        parameters="$parameters -showFactor $show_factor"
fi
if [ $help == "true"  ]; then
        parameters="$parameters -help"
fi

echo "  Hecho"                                                         | tee -a $report_file
echo ""                                                              | tee -a $report_file
echo "+-------------------------------------------------------------+"  | tee -a $report_file
```

```
#Crear el directorio donde se guardaran las imagenes preprocesadas
echo "Creando el directorio de imagenes preprocesadas..."              | tee -a $report_file
echo ""                                                                | tee -a $report_file
if [ -d $preprocessed_images_directory ]; then
    echo "  El directorio seleccionado ya existe"                      | tee -a $report_file
else
        mkdir $preprocessed_images_directory
    echo "  Hecho"                                                     | tee -a $report_file
fi
echo ""                                                                | tee -a $report_file
echo "+-------------------------------------------------------------+" | tee -a $report_file

#Ejecutar la aplicacion
echo "Ejecutando preprocessimages..."                                  | tee -a $report_file
echo ""                                                                | tee -a $report_file
start_time="$(date +%s)"
application $parameters                                        2>&1 | tee -a $report_file
end_time="$(date +%s)"
echo ""                                                                | tee -a $report_file
echo "+-------------------------------------------------------------+" | tee -a $report_file

#Calcular el tiempo de ejecucion
echo "Tiempo de ejecucion..."                                          | tee -a $report_file
echo ""                                                                | tee -a $report_file
exec_time=$(($end_time - $start_time))
exec_time_sec=$((($exec_time % 3600) % 60))
exec_time_min=$((($exec_time % 3600) / 60))
exec_time_hour=$(($exec_time / 3600))
echo "  "$exec_time_hour"h "$exec_time_min"m "$exec_time_sec"s"       | tee -a $report_file
echo ""| tee -a $report_file
echo "+-------------------------------------------------------------+" | tee -a $report_file

#Pausa
echo "Pulse enter para salir..."                                       | tee -a $report_file
echo ""                                                                | tee -a $report_file
read -n 0 -ers

#Salir
exit
```

5.4. *script_createsamples.sh*

```
#!/bin/bash

#Establecer la ruta de la aplicacion "opencv_createsamples"
application="opencv_createsamples"

#Definir el archivo donde se guardara el informe de "script_createsamples.sh"
report_file="../Informes/createsamples/informe.txt"

#Imprimir el titulo
echo "+-------------------------------------------------------------+"  | tee $report_file
echo "|                    SCRIPT_CREATESAMPLES                     |" | tee -a $report_file
echo "+-------------------------------------------------------------+"  | tee -a $report_file

#Cargar los parametros de la aplicacion
echo "Cargando parametros..."                                          | tee -a $report_file
echo ""                                                                | tee -a $report_file
                                                                       #<default>
collection_file_name="../Listas/createsamples/lista_positivas.txt"     #<null>
vec_file_name="../Muestras/muestras.vec"                               #<null>
number_of_samples=1000                                                 #<1000>
sample_width=24                                                        #<24>
sample_height=24                                                       #<24>

parameters= "-info $collection_file_name -vec $vec_file_name -num $number_of_samples -w
$sample_width -h $sample_height"

echo "  Hecho"                                                         | tee -a $report_file
echo ""                                                                | tee -a $report_file
echo "+-------------------------------------------------------------+"  | tee -a $report_file

#Ejecutar la aplicacion
echo "Ejecutando opencv_createsamples..."                              | tee -a $report_file
echo ""                                                                | tee -a $report_file
start_time="$(date +%s)"
application $parameters                                        2>&1 | tee -a $report_file
end_time="$(date +%s)"
echo ""                                                                | tee -a $report_file
echo "+-------------------------------------------------------------+"  | tee -a $report_file

#Calcular el tiempo de ejecucion
echo "Tiempo de ejecucion..."                                          | tee -a $report_file
echo ""                                                                | tee -a $report_file
exec_time=$(($end_time - $start_time))
exec_time_sec=$((($exec_time % 3600) % 60))
exec_time_min=$((($exec_time % 3600) / 60))
exec_time_hour=$(($exec_time / 3600))
echo "  "$exec_time_hour"h "$exec_time_min"m "$exec_time_sec"s"         | tee -a $report_file
echo ""                                                                | tee -a $report_file
echo "+-------------------------------------------------------------+"  | tee -a $report_file

#Pausa
echo "Pulse enter para salir..."                                       | tee -a $report_file
echo ""                                                                | tee -a $report_file
read -n 0 -ers

#Salir
exit
```

5.5. *script_showsamples.sh*

```
#!/bin/bash

#Establecer la ruta de la aplicacion "opencv_createsamples"
application="opencv_createsamples"

#Titulo
echo "+-------------------------------------------------------------+"
echo "|                      SCRIPT_SHOWSAMPLES                     |"
echo "+-------------------------------------------------------------+"

#Cargar los parametros de la aplicacion
echo "Cargando parametros..."
echo ""
                                                                    #<default>
vec_file_name="../Muestras/muestras.vec"                            #<null>
scale_factor=4                                                      #<4>
sample_width=24                                                     #<24>
sample_height=24                                                    #<24>

parameters="-vec $vec_file_name -show $scale_factor -w $sample_width -h $sample_height"

echo "Hecho"
echo ""
echo "+-------------------------------------------------------------+"

#Ejecutar la aplicacion
echo "Ejecutando opencv_createsamples..."
echo ""
$application $parameters
echo ""
echo "+-------------------------------------------------------------+"

#Pausa
echo "Pulse enter para salir..."
echo ""
read -n 0 -ers

#Salir
exit
```

5.6. *script_traincascade.sh*

```
#!/bin/bash

#Establecer la ruta de la aplicacion "opencv_traincascade"
application="opencv_traincascade"

#Definir el archivo donde se guardara el informe de "script_traincascade.sh"
report_file="../Informes/traincascade/informe.txt"

#Imprimir el titulo
echo "+-------------------------------------------------------------+"  | tee $report_file
echo "|                    SCRIPT_TRAINCASCADE                      |" | tee -a $report_file
echo "+-------------------------------------------------------------+"  | tee -a $report_file

#Cargar los parametros de la aplicacion
echo "Cargando parametros..."                                          | tee -a $report_file
echo ""                                                                | tee -a $report_file
                                                                       #<default>
cascade_dir_name="../Clasificadores/Clasificador"                      #<null>
vec_file_name="../Muestras/muestras.vec"                               #<null>
background_file_name="../Listas/traincascade/lista_negativas.txt"      #<null>
number_of_positive_samples=2000                                        #<2000>
percentage_of_positive_samples_used_per_stage=100                      #<100>
number_of_negative_samples=1000                                        #<1000>
number_of_stages=20                                                    #<20>
precalculated_vals_buffer_size_in_Mb=256                               #256>
precalculated_idxs_buffer_size_in_Mb=256                               #<256>
feature_type="HAAR"                                                    #<{ HAAR | LBP }>
sample_width=24                                                        #<24 (default)>
sample_height=24                                                       #<24 (default)>
boost_type="GAB"                                             #<{ DAB | RAB | LB | GAB (*) }>
min_hit_rate=0.995                                                     #<0.995>
max_false_alarm_rate=0.5                                               #<0.5>
weight_trim_rate=0.95                                                  #<0.95>
max_depth_of_weak_tree=1                                               #<1>
max_weak_tree_count=100                                                #<100>
haar_feature_mode="BASIC"                                    #<{BASIC (*) | CORE | ALL}>

echo "  Hecho"                                                         | tee -a $report_file
echo ""                                                              | tee -a $report_file
echo "+-------------------------------------------------------------+"  | tee -a $report_file

#Crear el directorio donde se guardara el clasificador
echo "Creando el directorio del clasificador..."                     | tee -a $report_file
echo ""                                                              | tee -a $report_file

if [ -d $cascade_dir_name ]; then
    echo "  El directorio seleccionado ya existe"                    | tee -a $report_file
else
        mkdir $cascade_dir_name
    echo "  Hecho"                                                   | tee -a $report_file
fi

echo ""                                                              | tee -a $report_file
echo "+-------------------------------------------------------------+"  | tee -a $report_file

#Calcular el numero de muestras positivas utilizadas por etapa
echo "Calculando el numero de muestras positivas por etapa..."       | tee -a $report_file
echo ""                                                              | tee -a $report_file
number_of_positive_samples_used_per_stage=$((($number_of_positive_samples *
$percentage_of_positive_samples_used_per_stage) / 100))
echo "  Hecho"                                                       | tee -a $report_file
echo ""                                                              | tee -a $report_file
echo "+-------------------------------------------------------------+"  | tee -a $report_file

#Establecer los parametros de la aplicacion
parameters="$cascade_dir_name -vec $vec_file_name -bg $background_file_name -numPos
$number_of_positive_samples_used_per_stage -numNeg $number_of_negative_samples -numStages
$number_of_stages -precalcValBufSize $precalculated_vals_buffer_size_in_Mb -
precalcIdxBufSize $precalculated_idxs_buffer_size_in_Mb -featureType $feature_type -w
$sample_width -h $sample_height -bt $boost_type -minHitRate $min_hit_rate -maxFalseAlarmRate
```

```
$max_false_alarm_rate -weightTrimRate $weight_trim_rate -maxDepth $max_depth_of_weak_tree -
maxWeakCount $max_weak_tree_count -mode $haar_feature_mode"

#Ejecutar la aplicacion
echo "Ejecutando opencv_traincascade..."                              | tee -a $report_file
echo ""                                                               | tee -a $report_file
start_time="$(date +%s)"
application $parameters                                          2>&1 | tee -a $report_file
end_time="$(date +%s)"
echo ""                                                               | tee -a $report_file
echo "+-------------------------------------------------------------+" | tee -a $report_file

#Calcular el tiempo de ejecucion
echo "Tiempo de ejecucion..."                                         | tee -a $report_file
echo ""                                                               | tee -a $report_file
exec_time=$(($end_time - $start_time))
exec_time_sec=$((($exec_time % 3600) % 60))
exec_time_min=$((($exec_time % 3600) / 60))
exec_time_hour=$(($exec_time / 3600))
echo "  "$exec_time_hour"h "$exec_time_min"m "$exec_time_sec"s"        | tee -a $report_file
echo ""                                                               | tee -a $report_file
echo "+-------------------------------------------------------------+" | tee -a $report_file

#Pausa
echo "Pulse enter para salir..."                                      | tee -a $report_file
echo ""                                                               | tee -a $report_file
read -n 0 -ers

#Salir
exit
```

5.7. *script_testcascade.sh*

```
#!/bin/bash

#Establecer la ruta de la aplicacion "testcascade"
application="../Aplicaciones/testcascade/build/testcascade"

#Definir el archivo donde se guardara el informe de "script_testcascade.sh"
report_file="../Informes/testcascade/informe.txt"

#Imprimir el titulo
echo "+-------------------------------------------------------------+"  | tee $report_file
echo "|                    SCRIPT_TESTCASCADE                       |" | tee -a $report_file
echo "+-------------------------------------------------------------+"  | tee -a $report_file

#Cargar los parametros de la aplicacion "testcascade"
echo "Cargando parametros..."                                          | tee -a $report_file
echo ""                                                                | tee -a $report_file
                                                                       #<default>
cascade_file_name="../Clasificadores/Clasificador/cascade.xml"         #<null>
list_file_name="../Listas/testcascade/lista_positivas.txt"             #<null>
data_file_name="../Curvas/curvas.txt"                                  #<null>
scale_factor=1.1                                                       #<1.1>
min_object_width=0                                                     #<0>
min_object_height=0                                                    #<0>
max_object_width=0                                                     #<0>
max_object_height=0                                                    #<0>
group_threshold=1                                                      #<1>
min_nms_overlap=0.5                                                    #<0.5>
min_match_overlap=0.5                                                  #<0.5 >
curve_type="all"                                    #<{ roc | det | pr | all (*) | null }>
show="false"                                                           #<false>
help="false"                                                           #<false>

parameters=""
if [ $cascade_file_name != "null" ]; then
        parameters="$parameters -cascade $cascade_file_name"
fi
if [ $list_file_name != "null" ]; then
        parameters="$parameters -list $list_file_name"
fi
if [ $data_file_name != "null" ]; then
        parameters="$parameters -data $data_file_name"
fi
if [ $scale_factor != "-1" ]; then
        parameters="$parameters -scale $scale_factor"
fi
if [ $min_object_width != "-1" ] && [ $min_object_height != "-1" ]; then
        parameters="$parameters -minsize $min_object_width $min_object_height"
fi
if [ $max_object_width != "-1" ] && [ $max_object_height != "-1" ]; then
        parameters="$parameters -maxsize $max_object_width $max_object_height"
fi
if [ $group_threshold != "-1" ]; then
        parameters="$parameters -group $group_threshold"
fi
if [ $min_nms_overlap != "-1" ]; then
        parameters="$parameters -noverlap $min_nms_overlap"
fi
if [ $min_match_overlap != "-1" ]; then
        parameters="$parameters -moverlap $min_match_overlap"
fi
if [ $curve_type != "null" ]; then
        parameters="$parameters -curve $curve_type"
fi
if [ $show == "true" ]; then
        parameters="$parameters -show"
fi
if [ $help == "true" ]; then
        parameters="$parameters -help"
fi
```

```
echo "  Hecho"                                                          | tee -a $report_file
echo ""                                                                 | tee -a $report_file
echo "+-------------------------------------------------------------+" | tee -a $report_file

#Ejecutar la aplicacion
echo "Ejecutando testcascade..."                                        | tee -a $report_file
echo ""                                                                 | tee -a $report_file
start_time="$(date +%s)"
application $parameters                                          2>&1 | tee -a $report_file
end_time="$(date +%s)"
echo ""                                                                 | tee -a $report_file
echo "+-------------------------------------------------------------+" | tee -a $report_file

#Calcular el tiempo de ejecucion
echo "Tiempo de ejecucion..."                                           | tee -a $report_file
echo ""                                                                 | tee -a $report_file
exec_time=$(($end_time - $start_time))
exec_time_sec=$((($exec_time % 3600) % 60))
exec_time_min=$((($exec_time % 3600) / 60))
exec_time_hour=$(($exec_time / 3600))
echo "  "$exec_time_hour"h "$exec_time_min"m "$exec_time_sec"s"        | tee -a $report_file
echo ""                                                                 | tee -a $report_file
echo "+-------------------------------------------------------------+" | tee -a $report_file

#Pausa
echo "Pulse enter para salir..."                                        | tee -a $report_file
echo ""                                                                 | tee -a $report_file
read -n 0 -ers

#Salir
exit
```

5.8. *tldr.cpp*

```
// ----------------------------------------------------------------------------------------0
// Includes and namespace

#include <stdlib.h>
#include <string>
#include <vector>
#include <iostream>
#include <fstream>

#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/features2d/features2d.hpp"

using namespace std;
using namespace cv;

// ----------------------------------------------------------------------------------------0
// Variable Types

enum slColor { red, amberup, green, amberdown, amberorgreen, amberupordown, unknown };

struct siParameters
{
    bool listInput;
    string listFileName ;
    ifstream listFile;

    bool videoInput;
    string videoFileName;

    bool cameraInput;
    int cameraNumber;

    int frameRate;
    VideoCapture videoCapture;
};

struct tldParameters
{
    string cascadeFileName;
    CascadeClassifier cascade;

    float shrinkFactor;
    float cropFactor;
    int blurSize;

    float scaleFactor;
    Size minSize;
    Size maxSize;

    int minGroup;
    float minOverlap;
    int maxNumber;
};

struct bdParameters
{
    int threshStep;
    int minThresh;
    int maxThresh;

    bool filterDiam;
    int minDiam;
    int maxDiam;

    bool filterCirc;
    float minCirc;
    float maxCirc;
```

```
    bool filterInert;
    float minInert;
    float maxInert;

    bool filterConvex;
    float minConvex;
    float maxConvex;
};

struct sldParameters
{
    int blurSize;

    bdParameters bdParams;

    int minGroup;
    float minOverlap;
};

struct pParameters
{
     int blurSize;
     int sobelSize;
     int edgesThresh;

     float boundFactor;
     float widthFactor;
     float distFactor;
};

struct mParameters
{
    vector<string> templateFileNames;
    vector<Mat> matchingTemplates;
    vector<Rect> lightBoxes;

    int blurSize;
};

struct hParameters
{
    float intHaloFactor;
    float extHaloFactor;

    int blurSize;

    float histFactor;

    int minRedHue;
    int maxRedHue;

    int minAmberHue;
    int maxAmberHue;

    int minGreenHue;
    int maxGreenHue;
};

struct slrParameters
{
    string recognType;
    pParameters pParams;
    mParameters mParams;
    hParameters hParams;
};

struct drdParameters
{
    vector<string> imageFileNames;
    vector<Mat>    recognitionImages;

    string drawColor;
    int drawThickness;
```

```
    float scaleFactor;
};

// ------------------------------------------------------------------------------------------0
// Functions

void initializeParameters( siParameters &siParams, tldParameters &tldParams,
                           sldParameters &sldParams, slrParameters &slrParameters,
                           drdParameters &drdParams, bool &usageHelp )
{
    // ------------------------------------------------------------------------1
    // Initialize program parameters

    // ---------------------------------------------------------------2
    // Source input parameters

    siParams.listInput = false;
    siParams.listFileName = "null";

    siParams.videoInput = false;
    siParams.videoFileName = "null";

    siParams.cameraInput = false;
    siParams.cameraNumber = 0;

    siParams.frameRate = 5;

    // ---------------------------------------------------------------2
    // Traffic lights detection parameters

    tldParams.cascadeFileName = "null";

    tldParams.shrinkFactor = 0.5f;
    tldParams.cropFactor = 0.5f;
    tldParams.blurSize = 3;

    tldParams.scaleFactor = 1.1f;
    tldParams.minSize = Size();
    tldParams.maxSize = Size();

    tldParams.minGroup = 1;
    tldParams.minOverlap = 0.5f;
    tldParams.maxNumber = 3;

    // ---------------------------------------------------------------2
    // Spot lights detection parameters

    sldParams.blurSize = 3;

    sldParams.bdParams.threshStep = 5;
    sldParams.bdParams.minThresh = 0;
    sldParams.bdParams.maxThresh = 254;

    sldParams.bdParams.filterDiam = true;
    sldParams.bdParams.minDiam = 10;
    sldParams.bdParams.maxDiam = 40;

    sldParams.bdParams.filterCirc = true;
    sldParams.bdParams.minCirc = 0.8f;
     sldParams.bdParams.maxCirc = 1.0f;

    sldParams.bdParams.filterInert = true;
    sldParams.bdParams.minInert = 0.8f;
    sldParams.bdParams.maxInert = 1.2f;

    sldParams.bdParams.filterConvex = true;
    sldParams.bdParams.minConvex = 0.8f;
    sldParams.bdParams.maxConvex = 1.2f;

    sldParams.minGroup = 1;
    sldParams.minOverlap = 0.5f;
```

```
    // ------------------------------------------------------------------2
    // Spot lights recognition parameters

    slrParams.recognType = "null";

    // - Spot lights recognition by position parameters

    slrParams.pParams.blurSize = 3;

    slrParams.pParams.sobelSize = 3;
    slrParams.pParams.edgesThresh = 100;

    slrParams.pParams.boundFactor = 0.25f;
    slrParams.pParams.widthFactor = 0.8f;
    slrParams.pParams.distFactor = 1.5f;

    // - Spot lights recognition by matching parameters

    slrParams.mParams.templateFileNames = vector <string> ( 3 );
    slrParams.mParams.templateFileNames[0] = "null";
    slrParams.mParams.templateFileNames[1] = "null";
    slrParams.mParams.templateFileNames[2] = "null";

    slrParams.mParams.lightBoxes = vector <Rect> ( 3 );
    slrParams.mParams.lightBoxes[0] = Rect( -1, -1, -1, -1 );
    slrParams.mParams.lightBoxes[1] = Rect( -1, -1, -1, -1 );
    slrParams.mParams.lightBoxes[2] = Rect( -1, -1, -1, -1 );

    slrParams.mParams.blurSize = 3;

    // - Spot lights recognition by hue parameters

    slrParams.hParams.blurSize = 3;

    slrParams.hParams.intHaloFactor = 0.25f;
    slrParams.hParams.extHaloFactor = 1.25f;

    slrParams.hParams.histFactor = 0.5f;

    slrParams.hParams.maxRedHue = 130;
    slrParams.hParams.minRedHue = 120;
    slrParams.hParams.maxAmberHue = 100;
    slrParams.hParams.minAmberHue = 90;
    slrParams.hParams.maxGreenHue = 40;
    slrParams.hParams.minGreenHue = 30;

    // ------------------------------------------------------------------2
    // Detection and recognition display parameters

    drdParams.imageFileNames = vector <string> ( 7 );
    drdParams.imageFileNames[0] = "null";
    drdParams.imageFileNames[1] = "null";
    drdParams.imageFileNames[2] = "null";
    drdParams.imageFileNames[3] = "null";
    drdParams.imageFileNames[4] = "null";
    drdParams.imageFileNames[5] = "null";
    drdParams.imageFileNames[6] = "null";

    drdParams.drawColor = "blue";
    drdParams.drawThickness = 4;

    drdParams.scaleFactor = 0.5f;

    // ------------------------------------------------------------------2
    // Usage help

    usageHelp = false;

}

int readParameters( int argc, char** argv, siParameters &siParams, tldParameters &tldParams,
                    sldParameters &sldParams, slrParameters &slrParams,
                    drdParameters &drdParams, bool &usageHelp )
```

```
{
    // ------------------------------------------------------------------------1
    // Read program parameters

    for( int i = 1; i < argc; i++ )
    {
        // ------------------------------------------------------------2
        // Souce input parameters

        if( !strcmp( argv[i], "-siListFileName" ) )
        {
            siParams.listInput = true;
            siParams.listFileName = argv[++i];
        }
        else if( !strcmp( argv[i], "-siVideoFileName" ) )
        {
            siParams.videoInput = true;
            siParams.videoFileName = argv[++i];
        }
        else if( !strcmp( argv[i], "-siCameraNumber" ) )
        {
            siParams.cameraInput = true;
            siParams.cameraNumber = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-siFrameRate" ) )
        {
            siParams.frameRate = atoi( argv[++i] );
        }

        // ------------------------------------------------------------2
        // Traffic lights detection parameters

        else if( !strcmp( argv[i], "-tldShrinkFactor" ) )
        {
            tldParams.shrinkFactor = float( atof( argv[++i] ) );
        }
        else if( !strcmp( argv[i], "-tldCropFactor" ) )
        {
            tldParams.cropFactor = float( atof( argv[++i] ) );
        }
        else if( !strcmp( argv[i], "-tldScaleFactor" ) )
        {
            tldParams.scaleFactor = float( atof( argv[++i] ) );
        }
        else if( !strcmp( argv[i], "-tldMinWidth" ) )
        {
            tldParams.minSize.width = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-tldMinHeight" ) )
        {
            tldParams.minSize.height = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-tldMaxWidth" ) )
        {
            tldParams.maxSize.width = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-tldMaxHeight" ) )
        {
            tldParams.maxSize.height = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-tldMinGroup" ) )
        {
            tldParams.minGroup = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-tldMinOverlap" ) )
        {
            tldParams.minOverlap = float( atof( argv[++i] ) );
        }
        else if( !strcmp( argv[i], "-tldMaxNumber" ) )
        {
            tldParams.maxNumber = atoi( argv[++i] );
        }
```

```
        // ----------------------------------------------------2
        // Spot lights detection parameters

        else if( !strcmp( argv[i], "-sldBlurSize" ) )
        {
            sldParams.blurSize = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-sldThreshStep" ) )
        {
            sldParams.bdParams.threshStep = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-sldMinThresh" ) )
        {
            sldParams.bdParams.minThresh = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-sldMaxThresh" ) )
        {
            sldParams.bdParams.maxThresh = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-sldMinDiam" ) )
        {
            sldParams.bdParams.minDiam = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-sldMaxDiam" ) )
        {
            sldParams.bdParams.maxDiam = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-sldMinCirc" ) )
        {
            sldParams.bdParams.minCirc = float( atof( argv[++i] ) );
        }
        else if( !strcmp( argv[i], "-sldMaxCirc" ) )
        {
            sldParams.bdParams.maxCirc = float( atof( argv[++i] ) );
        }
        else if( !strcmp( argv[i], "-sldMinInert" ) )
        {
            sldParams.bdParams.minInert = float( atof( argv[++i] ) );
        }
        else if( !strcmp( argv[i], "-sldMaxInert" ) )
        {
            sldParams.bdParams.maxInert = float( atof( argv[++i] ) );
        }
        else if( !strcmp( argv[i], "-sldMinConvex" ) )
        {
            sldParams.bdParams.minConvex = float( atof( argv[++i] ) );
        }
        else if( !strcmp( argv[i], "-sldMaxConvex" ) )
        {
            sldParams.bdParams.maxConvex = float( atof( argv[++i] ) );
        }
        else if( !strcmp( argv[i], "-sldMinGroup" ) )
        {
            sldParams.minGroup = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-sldMinOverlap" ) )
        {
            sldParams.minOverlap = float( atof( argv[++i] ) );
        }

        // ----------------------------------------------------------2
        // Spot lights recognition parameters

        else if( !strcmp( argv[i], "-recognType" ) )
        {
            slrParams.recognType = argv[++i];
        }

        // - Spot lights recognition by position parameters

        else if( !strcmp( argv[i], "-slrpBlurSize" ) )
        {
            slrParams.pParams.blurSize = atoi( argv[++i] );
```

```
        }
        else if( !strcmp( argv[i], "-slrpSobelSize" ) )
        {
            slrParams.pParams.sobelSize = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-slrpEdgesThresh" ) )
        {
            slrParams.pParams.edgesThresh = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-slrpBoundFactor" ) )
        {
            slrParams.pParams.boundFactor = float( atof( argv[++i] ) );
        }
        else if( !strcmp( argv[i], "-slrpWidthFactor" ) )
        {
            slrParams.pParams.widthFactor = float( atof( argv[++i] ) );
        }
        else if( !strcmp( argv[i], "-slrpDistFactor" ) )
        {
            slrParams.pParams.distFactor = float( atof( argv[++i] ) );
        }

        // - Spot lights recognition by matching parameters

        else if( !strcmp( argv[i], "-slrmRedTemplFileName" ) )
        {
            slrParams.mParams.templateFileNames[0] = argv[++i];
        }
        else if( !strcmp( argv[i], "-slrmAmberTemplFileName" ) )
        {
            slrParams.mParams.templateFileNames[1] = argv[++i];
        }
        else if( !strcmp( argv[i], "-slrmGreenTemplFileName" ) )
        {
            slrParams.mParams.templateFileNames[2] = argv[++i];
        }
        else if( !strcmp( argv[i], "-slrmRedLightBoxX" ) )
        {
            slrParams.mParams.lightBoxes[0].x = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-slrmRedLightBoxY" ) )
        {
            slrParams.mParams.lightBoxes[0].y = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-slrmRedLightBoxW" ) )
        {
            slrParams.mParams.lightBoxes[0].width = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-slrmRedLightBoxH" ) )
        {
            slrParams.mParams.lightBoxes[0].height = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-slrmAmberLightBoxX" ) )
        {
            slrParams.mParams.lightBoxes[1].x = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-slrmAmberLightBoxY" ) )
        {
            slrParams.mParams.lightBoxes[1].y = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-slrmAmberLightBoxW" ) )
        {
            slrParams.mParams.lightBoxes[1].width = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-slrmAmberLightBoxH" ) )
        {
            slrParams.mParams.lightBoxes[1].height = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-slrmGreenLightBoxX" ) )
        {
            slrParams.mParams.lightBoxes[2].x = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-slrmGreenLightBoxY" ) )
```

```
        {
            slrParams.mParams.lightBoxes[2].y = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-slrmGreenLightBoxW" ) )
        {
            slrParams.mParams.lightBoxes[2].width = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-slrmGreenLightBoxH" ) )
        {
            slrParams.mParams.lightBoxes[2].height = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-slrmBlurSize" ) )
        {
            slrParams.mParams.blurSize = atoi( argv[++i] );
        }

        // - Spot lights recognition by hue parameters

        else if( !strcmp( argv[i], "-slrhBlurSize" ) )
        {
            slrParams.hParams.blurSize = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-slrhIntHaloFactor" ) )
        {
            slrParams.hParams.intHaloFactor = float( atof( argv[++i] ) );
        }
        else if( !strcmp( argv[i], "-slrhExtHaloFactor" ) )
        {
            slrParams.hParams.extHaloFactor = float( atof( argv[++i] ) );
        }
        else if( !strcmp( argv[i], "-slrhHistFactor" ) )
        {
            slrParams.hParams.histFactor = float( atof( argv[++i] ) );
        }
        else if( !strcmp( argv[i], "-slrhMaxRedHue" ) )
        {
            slrParams.hParams.maxRedHue = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-slrhMinRedHue" ) )
        {
            slrParams.hParams.minRedHue = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-slrhMaxAmberHue" ) )
        {
            slrParams.hParams.maxAmberHue = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-slrhMinAmberHue" ) )
        {
            slrParams.hParams.minAmberHue = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-slrhMaxGreenHue" ) )
        {
            slrParams.hParams.maxGreenHue = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-slrhMinGreenHue" ) )
        {
            slrParams.hParams.minGreenHue = atoi( argv[++i] );
        }

        // ------------------------------------------------------------2
        // Detection and recognition display parameters

        else if( !strcmp( argv[i], "-drdRedImageFileName" ) )
        {
            drdParams.imageFileNames[0] = argv[++i];
        }
        else if( !strcmp( argv[i], "-drdAmberUpImageFileName" ) )
        {
            drdParams.imageFileNames[1] = argv[++i];
        }
        else if( !strcmp( argv[i], "-drdGreenImageFileName" ) )
        {
            drdParams.imageFileNames[2] = argv[++i];
```

```
        }
        else if( !strcmp( argv[i], "-drdAmberDownImageFileName" ) )
        {
            drdParams.imageFileNames[3] = argv[++i];
        }
        else if( !strcmp( argv[i], "-drdAmberOrGreenImageFileName" ) )
        {
            drdParams.imageFileNames[4] = argv[++i];
        }
        else if( !strcmp( argv[i], "-drdAmberUpOrDownImageFileName" ) )
        {
            drdParams.imageFileNames[5] = argv[++i];
        }
        else if( !strcmp( argv[i], "-drdUnknownImageFileName" ) )
        {
            drdParams.imageFileNames[6] = argv[++i];
        }
        else if( !strcmp( argv[i], "-drdDrawColor" ) )
        {
            drdParams.drawColor = argv[++i];
        }
        else if( !strcmp( argv[i], "-drdDrawThickness" ) )
        {
            drdParams.drawThickness = atoi( argv[++i] );
        }
        else if( !strcmp( argv[i], "-drdScaleFactor" ) )
        {
            drdParams.scaleFactor = float( atof( argv[++i] ) );
        }

        // ------------------------------------------------------------------2
        // Usage help

        else if( !strcmp( argv[i], "-help" ) )
        {
            usageHelp = true;
        }

        // ------------------------------------------------------------------2
        // Wrong parameter

        else
        {
            cerr << "Reading program parameters..." << endl
                 << endl
                 << "  Wrong parameter: " << argv[i] << endl
                 << endl;
            return 1;
        }

    }

    return 0;

}

void printUsageHelp()
{
    // ------------------------------------------------------------------------1
    // Print usage help

    // ------------------------------------------------------------------2
    // Souce input parameters

    cout << "  Source input parameters:" << endl
         << endl
         << "    [-siListFileName <list_file_name>]" << endl
         << "    [-siVideoFileName <video_file_name>]" << endl
         << "    [-siCameraNumber <camera_number>]" << endl
         << "    [-siFrameRate <frame_rate>]" << endl
         << endl;

    // ------------------------------------------------------------------2
```

```
    // Traffic lights detection parameters

    cout << "  Traffic lights detection parameters:"
         << endl
         << "    [-tldCascadeFileName <cascade_file_name>]" << endl
         << "    [-tldShrinkFactor <preprocess_shrink_factor>]" << endl
         << "    [-tldCropFactor <preprocess_crop_factor>]" << endl
         << "    [-tldBlurSize <preprocess_blur_kernel_size>]" << endl
         << "    [-tldScaleFactor <detection_scale_factor>]" << endl
         << "    [-tldMinWidth <detection_min_width>]" << endl
         << "    [-tldMinHeight <detection_min_height>]" << endl
         << "    [-tldMaxWidth <detection_max_width>]" << endl
         << "    [-tldMaxHeight <detection_max_height>]" << endl
         << "    [-tldMinGroup <min_bounding_boxes_group>]" << endl
         << "    [-tldMinOverlap <min_bounding_boxes_overlap>]" << endl
         << "    [-tldMaxNumber <max_bounding_boxes_number>]" << endl
         << endl;

    // ------------------------------------------------------------------2
    // Spot lights detection parameters

    cout << "  Spot lights detection parameters:"
         << endl
         << "    [-sldBlurSize <preprocess_blur_kernel_size>]" << endl
         << "    [-sldThreshStep <detection_threshold_step>]" << endl
         << "    [-sldMinThresh <detection_min_threshold>]" << endl
         << "    [-sldMaxThresh <detection_max_threshold>]" << endl
         << "    [-sldMinDiam <detection_min_diameter>]" << endl
         << "    [-sldMaxDiam <detection_max_diameter>]" << endl
         << "    [-sldMinCirc <detection_min_circularity>]" << endl
         << "    [-sldMaxCirc <detection_max_circularity>]" << endl
         << "    [-sldMinInert <detection_min_inertia_ratio>]" << endl
         << "    [-sldMaxInert <detection_max_inertia_ratio>]" << endl
         << "    [-sldMinConvex <detection_min_convexity>]" << endl
         << "    [-sldMaxConvex <detection_max_convexity>]" << endl
         << "    [-sldMinGroup <min_bounding_blobs_group>]" << endl
         << "    [-sldMinOverlap <min_bounding_blobs_overlap>]" << endl
         << endl;

    // ------------------------------------------------------------------2
    // Spot lights recognition parameters

    cout << "  Spot lights recognition parameters:"
         << endl
         << "    [-slrRecognType <recognition_type>]" << endl
         << endl;

    // - Spot lights recognition by position parameters

    cout << "  Spot lights recognition by position parameters:"
         << endl
         << "    [-slrpBlurSize <blur_kernel_size>]" << endl
         << "    [-slrpSobelSize <sobel_kernel_size>]" << endl
         << "    [-slrpEdgesThresh <sobel_edges_threshold>]" << endl
         << "    [-slrpBoundFactor <bound_lines_of_lights_factor>]" << endl
         << "    [-slrpWidthFactor <edges_width_factor>]" << endl
         << "    [-slrpDistFactor <distance_between_lights_factor>]" << endl
         << endl;

    // - Spot lights recognition by matching parameters

    cout << "  Spot lights recognition by matching parameters:"
         << endl
         << "    [-slrmRedTemplFileName <red_template_file_name>]" << endl
         << "    [-slrmAmberTemplFileName <amber_template_file_name>]" << endl
         << "    [-slrmGreenTemplFileName <green_template_file_name>]" << endl
         << "    [-slrmRedLightBoxX <red_light_box_x_location]" << endl
         << "    [-slrmRedLightBoxY <red_light_box_y_location]" << endl
         << "    [-slrmRedLightBoxW <red_light_box_width_size]" << endl
         << "    [-slrmRedLightBoxH <red_light_box_height_size]" << endl
         << "    [-slrmAmberLightBoxX <amber_light_box_x_location]" << endl
         << "    [-slrmAmberLightBoxY <amber_light_box_y_location]" << endl
         << "    [-slrmAmberLightBoxW <amber_light_box_width_size]" << endl
```

```
        << "    [-slrmAmberLightBoxH <amber_light_box_height_size]" << endl
        << "    [-slrmGreenLightBoxX <green_light_box_x_location]" << endl
        << "    [-slrmGreenLightBoxY <green_light_box_y_location]" << endl
        << "    [-slrmGreenLightBoxW <green_light_box_width_size]" << endl
        << "    [-slrmGreenLightBoxH <green_light_box_height_size]" << endl
        << "    [-slrmBlurSize <blur_kernel_size>]" << endl
        << endl;

    // - Spot lights recognition by hue parameters

    cout << "  Spot lights recognition by hue parameters:"
        << endl
        << "    [-slrhBlurSize <blur_kernel_size>]" << endl
        << "    [-slrhIntHaloFactor <internal_halo_factor>]" << endl
        << "    [-slrhExtHaloFactor <external_halo_factor>]" << endl
        << "    [-slrhHistFactor <max_amber_hue>]" << endl
        << "    [-slrhMaxRedHue <max_red_hue>]" << endl
        << "    [-slrhMinRedHue <min_red_hue>]" << endl
        << "    [-slrhMaxAmberHue <max_amber_hue>]" << endl
        << "    [-slrhMinAmberHue <min_amber_hue>]" << endl
        << "    [-slrhMaxGreenHue <max_green_hue>]" << endl
        << "    [-slrhMinGreenHue <min_green_hue>]" << endl
        << endl;

    // ------------------------------------------------------------------2
    // Display and recognition display parameters

    cout << "  Detection and recognition display:"
        << endl
        << "    [-drdRedImageFileName <red_image_file_name> ]" << endl
        << "    [-drdAmberImageUpFileName <amber_up_image_file_name> ]" << endl
        << "    [-drdGreenImageFileName <green_image_file_name> ]" << endl
        << "    [-drdAmberDownImageFileName <amber_down_image_file_name> ]" << endl
        << "    [-drdAmberOrGreenImageFileName <amber_or_green_image_file_name> ]" << endl
        << "    [-drdAmberUpOrDownImageFileName <amber_up_or_down_image_file_name> ]" << endl
        << "    [-drdUnknownImageFileName <unknown_image_file_name> ]" << endl
        << "    [-drdDrawColor <detection_draw_color>]" << endl
        << "    [-drdDrawThickness <detection_draw_thickness>]" << endl
        << "    [-drdScaleFactor <display_scale_factor>]" << endl
        << endl;

    // ------------------------------------------------------------------2
    // Help usage

    cout << "  Help usage:"
        << endl
        << "    [-help]" << endl
        << endl;

}

void printParameters( siParameters &siParams, tldParameters &tldParams,
                      sldParameters &sldParams, slrParameters &slrParams,
                      drdParameters &drdParams )
{
    // ----------------------------------------------------------------------------1
    // Print parameters

    // ------------------------------------------------------------------2
    // Souce input parameters

    cout << "  Source input parameters:" << endl
        << endl
        << "    List file: " << siParams.listFileName << endl
        << "    Video file: " << siParams.videoFileName << endl
        << "    Camera number: " << siParams.cameraNumber << endl
        << "    Frame rate: " << siParams.frameRate << endl
        << endl;

    // ------------------------------------------------------------------2
    // Traffic lights detection parameters

    cout << "  Traffic lights detection parameters:" << endl
```

```
        << endl
        << "    Cascade file: " << tldParams.cascadeFileName << endl
        << "    Shrink factor: " << tldParams.shrinkFactor << endl
        << "    Crop factor: " << tldParams.cropFactor << endl
        << "    Blur size: " << tldParams.blurSize << endl
        << "    Scale factor: " << tldParams.scaleFactor << endl
        << "    Min width: " << tldParams.minSize.width << endl
        << "    Min height: " << tldParams.minSize.height << endl
        << "    Max width: " << tldParams.maxSize.width << endl
        << "    Max height: " << tldParams.maxSize.height << endl
        << "    Min group: " << tldParams.minGroup << endl
        << "    Min overlap: " << tldParams.minOverlap << endl
        << "    Max number: " << tldParams.maxNumber << endl
        << endl;

    // ----------------------------------------------------------------2
    // Spot lights detection parameters

    cout << "  Spot lights detection parameters:" << endl
        << endl
        << "    Blur size: " << sldParams.blurSize << endl
        << "    Threshold step: " << sldParams.bdParams.threshStep << endl
        << "    Min threshold: " << sldParams.bdParams.minThresh << endl
        << "    Max threshold: " << sldParams.bdParams.maxThresh << endl
        << "    Min diameter: " << sldParams.bdParams.minDiam << endl
        << "    Max diameter: " << sldParams.bdParams.maxDiam << endl
        << "    Min circularity: " << sldParams.bdParams.minCirc << endl
        << "    Max circularity: " << sldParams.bdParams.maxCirc << endl
        << "    Min inertia ratio: " << sldParams.bdParams.minInert << endl
        << "    Max inertia ratio: " << sldParams.bdParams.maxInert << endl
        << "    Min convexity: " << sldParams.bdParams.minConvex << endl
        << "    Max convexity: " << sldParams.bdParams.maxConvex << endl
        << "    Min group: " << sldParams.minGroup << endl
        << "    Min overlap: " << sldParams.minOverlap << endl
        << endl;

    // ----------------------------------------------------------------2
    // Spot lights recognition parameters

    cout << "  Spot lights recognition parameters:" << endl
        << endl
        << "    Recognition type: " << slrParams.recognType << endl
        << endl;

    // - Spot lights recognition by position parameters

    cout << "  Spot lights recognition by position parameters:" << endl
        << endl
        << "    Blur size: " << slrParams.pParams.blurSize << endl
        << "    Sobel size: " << slrParams.pParams.sobelSize << endl
        << "    Edges trhesh: " << slrParams.pParams.edgesThresh << endl
        << "    Bound factor: " << slrParams.pParams.boundFactor << endl
        << "    Width factor: " << slrParams.pParams.widthFactor << endl
        << "    Distance factor: " << slrParams.pParams.distFactor << endl
        << endl;

    // - Spot lights recognition by matching parameters

    cout << "  Spot lights recognition by matching parameters:" << endl
        << endl
        << "    Red template file: " << slrParams.mParams.templateFileNames[0] << endl
        << "    Amber template file: " << slrParams.mParams.templateFileNames[1] << endl
        << "    Green template file: " << slrParams.mParams.templateFileNames[2] << endl
        << "    Red light box x: " << slrParams.mParams.lightBoxes[0].x << endl
        << "    Red light box y: " << slrParams.mParams.lightBoxes[0].y << endl
        << "    Red light box width: " << slrParams.mParams.lightBoxes[0].width << endl
        << "    Red light box height: " << slrParams.mParams.lightBoxes[0].height << endl
        << "    Amber light box x: " << slrParams.mParams.lightBoxes[1].x << endl
        << "    Amber light box y: " << slrParams.mParams.lightBoxes[1].y << endl
        << "    Amber light box width: " << slrParams.mParams.lightBoxes[1].width << endl
        << "    Amber light box height: " << slrParams.mParams.lightBoxes[1].height << endl
        << "    Green light box x: " << slrParams.mParams.lightBoxes[2].x << endl
        << "    Green light box y: " << slrParams.mParams.lightBoxes[2].y << endl
```

```
        << "    Green light box width: " << slrParams.mParams.lightBoxes[2].width << endl
        << "    Green light box height: " << slrParams.mParams.lightBoxes[2].height << endl
        << "    Blur size: " << slrParams.mParams.blurSize << endl
        << endl;

    // - Spot lights recognition by hue parameters

    cout << "  Spot lights recognition by hue parameters:" << endl
        << endl
        << "    Blur size: " << slrParams.hParams.blurSize << endl
        << "    Internal halo factor: " << slrParams.hParams.intHaloFactor << endl
        << "    External halo factor: " << slrParams.hParams.extHaloFactor << endl
        << "    Histogram factor: " << slrParams.hParams.histFactor << endl
        << "    Max red hue: " << slrParams.hParams.maxRedHue << endl
        << "    Min red hue: " << slrParams.hParams.minRedHue << endl
        << "    Max amber hue: " << slrParams.hParams.maxAmberHue << endl
        << "    Min amber hue: " << slrParams.hParams.minAmberHue << endl
        << "    Max green hue: " << slrParams.hParams.maxGreenHue << endl
        << "    Min green hue: " << slrParams.hParams.minGreenHue << endl
        << endl;

    // ------------------------------------------------------------------2
    // Display and recognition display parameters

    cout << "  Detection and recognition display:" << endl
        << endl
        << "    Red image file: " << drdParams.imageFileNames[0] << endl
        << "    Amber image file: " << drdParams.imageFileNames[1] << endl
        << "    Green image file: " << drdParams.imageFileNames[2] << endl
        << "    Amber down image file: " << drdParams.imageFileNames[3] << endl
        << "    Amber or green image file: " << drdParams.imageFileNames[4] << endl
        << "    Amber up or down image file: " << drdParams.imageFileNames[5] << endl
        << "    Unknown image file: " << drdParams.imageFileNames[6] << endl
        << "    Draw color: " << drdParams.drawColor << endl
        << "    Draw thickness: " << drdParams.drawThickness << endl
        << "    Scale factor: " << drdParams.scaleFactor << endl
        << endl;

}

int checkParameters( siParameters &siParams, tldParameters &tldParams,
                     sldParameters &sldParams, slrParameters &slrParams,
                     drdParameters &drdParams )
{
    // ------------------------------------------------------------------------1
    // Check program parameters

    // ------------------------------------------------------------------2
    // Source input parameters

    // - List file name

    if( siParams.listInput == true && siParams.listFileName == "null" )
    {
        cerr << "  In source input parameters." << endl
             << "  Invalid images list file name: " << siParams.listFileName << endl
             << endl;
        return 1;
    }

    // - Video file name

    if( siParams.videoInput == true && siParams.videoFileName == "null" )
    {
        cerr << "  In source input parameters." << endl
             << "  Invalid video file name: " << siParams.videoFileName << endl
             << endl;
        return 1;
    }

    // - Camera file name

    if( siParams.cameraInput == true && siParams.cameraNumber < 0 )
```

```
    {
       cerr << "  In source input parameters." << endl
            << "  Invalid video camera number: " << siParams.cameraNumber << endl
            << endl;
       return 1;
    }

    // - Frame Rate

    if( ( siParams.videoInput == true || siParams.cameraInput == true) &&
          siParams.frameRate <= 0 )
    {
       cerr << "  In source input parameters." << endl
            << "  Invalid frame rate: " << siParams.frameRate << endl
            << endl;
       return 1;
    }

    // ------------------------------------------------------------------2
    // Traffic lights detection parameters

    // - Cascade file name

    if( tldParams.cascadeFileName == "null" )
    {
       cerr << "  In traffic lights detection parameters." << endl
            << "  Invalid cascade file name: " << tldParams.cascadeFileName << endl
            << endl;
       return 1;
    }

    // - Shrink factor

    if( tldParams.shrinkFactor <= 0.0f || tldParams.shrinkFactor > 1.0f )
    {
       cerr << "  In traffic lights detection parameters." << endl
            << "  Invalid shrink factor: " << tldParams.shrinkFactor << endl
            << endl;
       return 1;
    }

    // - Crop factor

    if( tldParams.cropFactor <= 0.0f || tldParams.cropFactor > 1.0f )

    {
       cerr << "  In traffic lights detection parameters." << endl
            << "  Invalid crop factor: " << tldParams.cropFactor << endl
            << endl;
       return 1;
    }

    // - Blur size

    if( tldParams.blurSize < 3 || ( tldParams.blurSize - 1 ) % 2 )
    {
       cerr << "  In traffic lights detection parameters." << endl
            << "  Invalid blur size: " << tldParams.blurSize << endl
            << endl;
       return 1;
    }

    // - Scale factor

    if( tldParams.scaleFactor <= 1.0f )
    {
       cerr << "  In traffic lights detection parameters." << endl
            << "  Invalid scale factor: " << tldParams.scaleFactor << endl
            << endl;
       return 1;
    }

    // - Min width
```

```
if( tldParams.minSize.width < 0 )
{
   cerr << "  In traffic lights detection parameters." << endl
        << "  Invalid min size width: " << tldParams.minSize.width << endl
        << endl;
   return 1;
}

// - Min height

if( tldParams.minSize.height < 0 )
{
   cerr << "  In traffic lights detection parameters." << endl
        << "  Invalid min height: " << tldParams.minSize.height << endl
        << endl;           return 1;
}

// - Max width

if( tldParams.maxSize.width < 0 )
{
   cerr << "  In traffic lights detection parameters." << endl
        << "  Invalid max width: " << tldParams.maxSize.width << endl
        << endl;
   return 1;
}

// - Max height

if( tldParams.maxSize.height < 0 )
{
   cerr << "  In traffic lights detection parameters." << endl
        << "  Invalid max height: " << tldParams.maxSize.height << endl
        << endl;
   return 1;
}

// - Min-Max width

if( tldParams.minSize.width > tldParams.maxSize.width )
{
   cerr << "  In traffic lights detection parameters." << endl
        << "  Min width can not be greater than max width: "
        << tldParams.minSize.width << " > " << tldParams.maxSize.width << endl
        << endl;
   return 1;
}

// - Min-Max height

if( tldParams.minSize.height > tldParams.maxSize.height )
{
   cerr << "  In traffic lights detection parameters." << endl
        << "  Min height can not be greater than max v: "
        << tldParams.minSize.height << " > " << tldParams.maxSize.height << endl
        << endl;
   return 1;
}

// - Min group

if( tldParams.minGroup < 1 )
{
   cerr << "  In traffic lights detection parameters." << endl
        << "  Invalid min group: " << tldParams.minGroup << endl
        << endl;
   return 1;
}

// - Min overlap

if( tldParams.minOverlap < 0.0f || tldParams.minOverlap > 1.0f )
```

```
    {
       cerr << "  In traffic lights detection parameters." << endl
            << "  Invalid min overlap: " << tldParams.minOverlap << endl
            << endl;
       return 1;
    }

    // - Max number

    if( tldParams.maxNumber < 1 )
    {
       cerr << "  In traffic lights detection parameters." << endl
            << "  Invalid max number: " << tldParams.maxNumber << endl
            << endl;
       return 1;
    }

    // ------------------------------------------------------------------2
    // Spot lights detection parameters

    // - Blur size

    if( sldParams.blurSize < 3 || ( sldParams.blurSize - 1 ) % 2 )
    {
       cerr << "  In spot lights detection parameters." << endl
            << "  Invalid blur size: " << sldParams.blurSize << endl
            << endl;
       return 1;
    }

    // - Threshold step

    if( sldParams.bdParams.threshStep < 1 || sldParams.bdParams.threshStep > 254 )
    {
       cerr << "  In spot lights detection parameters." << endl
            << "  Invalid threshold step: " << sldParams.bdParams.threshStep << endl
            << endl;
       return 1;
    }

    // - Min threshold

    if( sldParams.bdParams.minThresh < 0 || sldParams.bdParams.minThresh > 254 )
    {
       cerr << "  In spot lights detection parameters." << endl
            << "  Invalid min threshold: " << sldParams.bdParams.minThresh << endl
            << endl;
       return 1;
    }

    // - Max threshold

    if( sldParams.bdParams.maxThresh < 0 || sldParams.bdParams.maxThresh > 254 )
    {
       cerr << "  In spot lights detection parameters." << endl
            << "  Invalid max threshold: " << sldParams.bdParams.maxThresh << endl
            << endl;
       return 1;
    }

    // - Min-Max threshold

    if( sldParams.bdParams.minThresh > sldParams.bdParams.maxThresh )
    {
       cerr << "  In spot lights detection parameters." << endl
            << "  Min threshold can not be greater than max threshold: "
            << sldParams.bdParams.minThresh << " > " << sldParams.bdParams.maxThresh<< endl
            << endl;
       return 1;
    }

    // - Min diameter
```

```
    if( sldParams.bdParams.minDiam < 1 )
    {
       cerr << "  In spot lights detection parameters." << endl
            << "  Invalid min diameter: " << sldParams.bdParams.minDiam << endl
            << endl;
       return 1;
    }

    // - Max diameter

    if( sldParams.bdParams.maxDiam < 1 )
    {
       cerr << "  In spot lights detection parameters." << endl
            << "  Invalid max diameter: " << sldParams.bdParams.maxDiam << endl
            << endl;
       return 1;
    }

    // - Min-Max diameter

    if( sldParams.bdParams.minDiam > sldParams.bdParams.maxDiam )
    {
       cerr << "  In spot lights detection parameters." << endl
            << "  Min diameter can not be greater than max diameter: "
            << sldParams.bdParams.minDiam << " > " << sldParams.bdParams.maxDiam << endl
            << endl;
       return 1;
    }

    // - Min circularity

    if( sldParams.bdParams.minCirc <= 0.0f )
    {
       cerr << "  In spot lights detection parameters." << endl
            << "  Invalid min circularity: " << sldParams.bdParams.minCirc << endl
            << endl;
       return 1;
    }

    // - Max circularity

    if( sldParams.bdParams.maxCirc <= 0.0f )
    {
       cerr << "  In spot lights detection parameters." << endl
            << "  Invalid max circularity: " << sldParams.bdParams.maxCirc << endl
            << endl;
       return 1;
    }

    // - Min-Max circularity

    if( sldParams.bdParams.minCirc > sldParams.bdParams.maxCirc )
    {
       cerr << "  In spot lights detection parameters." << endl
            << "  Min circularity can not be greater than max circularity: "
            << sldParams.bdParams.minCirc << " > " << sldParams.bdParams.maxCirc << endl
            << endl;
       return 1;
    }

    // - Min inertia ratio

    if( sldParams.bdParams.minInert <= 0.0f )
    {
       cerr << "  In spot lights detection parameters." << endl
            << "  Invalid min inertia ratio: " << sldParams.bdParams.minInert << endl
            << endl;
       return 1;
    }

    // - Max inertia ratio

    if( sldParams.bdParams.maxInert <= 0.0f )
```

```
    {
       cerr << "  In spot lights detection parameters." << endl
            << "  Invalid max inertia ratio: " << sldParams.bdParams.maxInert << endl
            << endl;
       return 1;
    }

    // - Min-Max inertia ratio

    if( sldParams.bdParams.minInert > sldParams.bdParams.maxInert )
    {
       cerr << "  In spot lights detection parameters." << endl
            << "  Min inertia ratio can not be greater than max inertia ratio: "
            << sldParams.bdParams.minInert << " > " << sldParams.bdParams.maxInert << endl
            << endl;
       return 1;
    }

    // - Min convexity

    if( sldParams.bdParams.minConvex <= 0.0f )
    {
       cerr << "  In spot lights detection parameters." << endl
            << "  Invalid min convexity: " << sldParams.bdParams.minConvex << endl
            << endl;
       return 1;
    }

    // - Max convexity

    if( sldParams.bdParams.maxConvex <= 0.0f )
    {
       cerr << "  In spot lights detection parameters." << endl
            << "  Invalid max convexity: " << sldParams.bdParams.maxConvex << endl
            << endl;
       return 1;
    }

    // - Min-Max convexity

    if( sldParams.bdParams.minConvex > sldParams.bdParams.maxConvex )
    {
       cerr << "  In spot lights detection parameters." << endl
            << "  Min convexity can not be greater than max convexity: "
            << sldParams.bdParams.minConvex << " > " << sldParams.bdParams.maxConvex << endl
            << endl;
       return 1;
    }

    // - Min group

    if( sldParams.minGroup < 1 )
    {
       cerr << "  In spot lights detection parameters." << endl
            << "  Invalid min group: " << sldParams.minGroup << endl
            << endl;
       return 1;
    }

    // ) Min overlap

    if( sldParams.minOverlap < 0.0f || sldParams.minOverlap > 1.0f )
    {
       cerr << "  In spot lights detection parameters." << endl
            << "  Invalid min overlap: " << sldParams.minOverlap << endl
            << endl;
       return 1;
    }

    // -----------------------------------------------------------------2
    // Spot lights recognition parameters

    // ---------------------------------------------------------3
```

```
    // Recognition type

    if( slrParams.recognType != "pos" && slrParams.recognType != "match" &&
slrParams.recognType != "hue" &&
        slrParams.recognType != "pos-hue" && slrParams.recognType != "match-hue" )
    {
        cerr << "  In spot lights recognition parameters." << endl
             << "  Invalid recognition type: " << slrParams.recognType << endl
             << endl;
        return 1;
    }

    // ------------------------------------------------------3
    // Spot lights recognition by position parameters

    if( slrParams.recognType == "pos" || slrParams.recognType == "pos-hue" )
    {
        // - Blur size

        if( slrParams.pParams.blurSize < 3 || ( slrParams.pParams.blurSize - 1 ) % 2 )
        {
            cerr << "  In spot lights recognition by position parameters." << endl
                 << "  Invalid blur size: " << slrParams.pParams.blurSize << endl
                 << endl;
            return 1;
        }

        // - Sobel size

        if( slrParams.pParams.sobelSize < 3 || ( slrParams.pParams.sobelSize - 1 ) % 2 )
        {
            cerr << "  In spot lights recognition by position parameters." << endl
                 << "  Invalid sobel size: " << slrParams.pParams.sobelSize << endl
                 << endl;
            return 1;
        }

        // - Edges threshold

        if( slrParams.pParams.edgesThresh < 0 || slrParams.pParams.edgesThresh > 254 )
        {
            cerr << "  In spot lights recognition by position parameters." << endl
                 << "  Invalid edges threshold: " << slrParams.pParams.edgesThresh << endl
                 << endl;
            return 1;
        }

        // - Bounding factor

        if( slrParams.pParams.boundFactor <= 0.0f || slrParams.pParams.boundFactor >= 1.0f )
        {
            cerr << "  In spot lights recognition by position parameters." << endl
                 << "  Invalid bounding factor threshold: " << slrParams.pParams.boundFactor
                 << endl << endl;
            return 1;
        }

         // - Width factor

        if( slrParams.pParams.widthFactor <= 0.0f || slrParams.pParams.widthFactor >= 1.0f )
        {
            cerr << "  In spot lights recognition by position parameters." << endl
                 << "  Invalid width factor: " << slrParams.pParams.widthFactor << endl
                 << endl;
            return 1;
        }

        // - Distance factor

        if( slrParams.pParams.distFactor < 1.0f || slrParams.pParams.distFactor >= 5.0f )
        {
            cerr << "  In spot lights recognition by position parameters." << endl
                 << "  Invalid distance factor: " << slrParams.pParams.distFactor << endl
```

```
            << endl;
        return 1;
    }

}

// ------------------------------------------------------3
// Spot lights recognition by matching parameters

if( slrParams.recognType == "match" || slrParams.recognType == "match-hue" )
{
    // - Red template file name

    if( slrParams.mParams.templateFileNames[0] == "null" )
    {
        cerr << "  In spot lights recognition by matching parameters." << endl
             << "  Invalid red template file name: "
             << slrParams.mParams.templateFileNames[0] << endl
             << endl;
        return 1;
    }

    // - Amber template file name

    if( slrParams.mParams.templateFileNames[1] == "null" )
    {
        cerr << "  In spot lights recognition by matching parameters." << endl
             << "  Invalid amber template file name: "
             << slrParams.mParams.templateFileNames[1] << endl
             << endl;
        return 1;
    }

    // - Green template file name

    if( slrParams.mParams.templateFileNames[2] == "null" )
    {
        cerr << "  In spot lights recognition by matching parameters." << endl
             << "  Invalid green template file name: "
             << slrParams.mParams.templateFileNames[2] << endl
             << endl;
        return 1;
    }

    // - Red light box x location

    if( slrParams.mParams.lightBoxes[0].x < 0 )
    {
        cerr << "  In spot lights recognition by matching parameters." << endl
             << "  Invalid red light box x location: "
             << slrParams.mParams.lightBoxes[0].x << endl
             << endl;
        return 1;
    }

    // - Red light box y location

    if( slrParams.mParams.lightBoxes[0].y < 0 )
    {
        cerr << "  In spot lights recognition by matching parameters." << endl
             << "  Invalid red light box y location: "
             << slrParams.mParams.lightBoxes[0].y << endl
             << endl;
        return 1;
    }

    // - Red light width size

    if( slrParams.mParams.lightBoxes[0].width <= 0 )
    {
        cerr << "  In spot lights recognition by matching parameters." << endl
             << "  Invalid red light box width size: "
             << slrParams.mParams.lightBoxes[0].width << endl
```

```
            << endl;
        return 1;
    }

    // - Red light height size

    if( slrParams.mParams.lightBoxes[0].height <= 0 )
    {
        cerr << "  In spot lights recognition by matching parameters." << endl
             << "  Invalid red light box height size: "
             << slrParams.mParams.lightBoxes[0].height << endl
             << endl;
        return 1;
    }

    // - Amber light box x location

    if( slrParams.mParams.lightBoxes[1].x < 0 )
    {
        cerr << "  In spot lights recognition by matching parameters." << endl
             << "  Invalid amber light box x location: "
             << slrParams.mParams.lightBoxes[1].x << endl
             << endl;
        return 1;
    }

    // - Amber light box y location

    if( slrParams.mParams.lightBoxes[1].y < 0 )
    {
        cerr << "  In spot lights recognition by matching parameters." << endl
             << "  Invalid amber light box y location: "
             << slrParams.mParams.lightBoxes[1].y << endl
             << endl;
        return 1;
    }

    // - Amber light width size

    if( slrParams.mParams.lightBoxes[1].width <= 0 )
    {
        cerr << "  In spot lights recognition by matching parameters." << endl
             << "  Invalid amber light box width size: "
             << slrParams.mParams.lightBoxes[1].width << endl
             << endl;
        return 1;
    }

    // - Amber light height size

    if( slrParams.mParams.lightBoxes[1].height <= 0 )
    {
        cerr << "  In spot lights recognition by matching parameters." << endl
             << "  Invalid amber light box height size: "
             << slrParams.mParams.lightBoxes[1].height << endl
             << endl;
        return 1;
    }

    // - Green light box x location

    if( slrParams.mParams.lightBoxes[2].x < 0 )
    {
        cerr << "  In spot lights recognition by matching parameters." << endl
             << "  Invalid green light box x location: "
             << slrParams.mParams.lightBoxes[2].x << endl
             << endl;
        return 1;
    }

    // - Green light box y location

    if( slrParams.mParams.lightBoxes[2].y < 0 )
```

```
        {
            cerr << "  In spot lights recognition by matching parameters." << endl
                 << "  Invalid green light box y location: "
                 << slrParams.mParams.lightBoxes[2].y << endl
                 << endl;
            return 1;
        }

        // - Green light width size

        if( slrParams.mParams.lightBoxes[2].width <= 0 )
        {
            cerr << "  In spot lights recognition by matching parameters." << endl
                 << "  Invalid green light box width size: "
                 << slrParams.mParams.lightBoxes[2].width << endl
                 << endl;
            return 1;
        }

        // - Green light height size

        if( slrParams.mParams.lightBoxes[2].height <= 0 )
        {
            cerr << "  In spot lights recognition by matching parameters." << endl
                 << "  Invalid green light box height size: "
                 << slrParams.mParams.lightBoxes[2].height << endl
                 << endl;
            return 1;
        }

        // - Blur size

        if( slrParams.mParams.blurSize < 3 || ( slrParams.mParams.blurSize - 1 ) % 2 )
        {
            cerr << "  In spot lights recognition by matching parameters." << endl
                 << "  Invalid blur size: " << slrParams.mParams.blurSize << endl
                 << endl;
            return 1;
        }

    }

    // -------------------------------------------------------3
    // Spot lights recognition by hue parameters

    if( slrParams.recognType == "hue" || slrParams.recognType == "pos-hue" ||
        slrParams.recognType == "match-hue" )
    {
        // - Internal halo factor

        if( slrParams.hParams.intHaloFactor < 0.0f || slrParams.hParams.intHaloFactor >= 1.0f)
        {
            cerr << "  In spot lights recognition by hue parameters." << endl
                 << "  Invalid internal halo factor: " << slrParams.hParams.intHaloFactor
                 << endl << endl;
            return 1;
        }

        // - External halo factor

        if( slrParams.hParams.extHaloFactor <= 1.0f || slrParams.hParams.extHaloFactor > 2.0f)
        {
            cerr << "  In spot lights recognition by hue parameters." << endl
                 << "  Invalid external halo factor: " << slrParams.hParams.extHaloFactor
                 << endl << endl;
            return 1;
        }

        // - Blur size

        if( slrParams.hParams.blurSize < 3 || ( slrParams.hParams.blurSize - 1 ) % 2 )
        {
            cerr << "  In spot lights recognition by hue parameters." << endl
```

```
            << "  Invalid blur size: " << slrParams.hParams.blurSize << endl
            << endl;
        return 1;
    }

    // - Histogram factor

    if( slrParams.hParams.histFactor < 0.0f || slrParams.hParams.histFactor > 1.0f )
    {
        cerr << "  In spot lights recognition by hue parameters." << endl
             << "  Invalid histogram factor: " << slrParams.hParams.histFactor << endl
             << endl;
        return 1;
    }

    // - Min red hue

    if( slrParams.hParams.minRedHue < 0 || slrParams.hParams.minRedHue > 180 )
    {
        cerr << "  In spot lights recognition by hue parameters." << endl
             << "  Invalid min red hue: " << slrParams.hParams.minRedHue << endl
             << endl;
        return 1;
    }

    // - Max red hue

    if( slrParams.hParams.maxRedHue < 0 || slrParams.hParams.maxRedHue > 180 )
    {
        cerr << "  In spot lights recognition by hue parameters." << endl
             << "  Invalid max red hue: " << slrParams.hParams.maxRedHue << endl
             << endl;
        return 1;
    }

    // - Min-Max red hue

    if( slrParams.hParams.minRedHue > slrParams.hParams.maxRedHue )
    {
        cerr << "  In spot lights recognition by hue parameters." << endl
             << "  Min red hue can not be greater than max red hue: "
             << slrParams.hParams.minRedHue << " > " << slrParams.hParams.maxRedHue << endl
             << endl;
        return 1;
    }

    // - Min red hue and max amber hue

    if( slrParams.hParams.minRedHue <= slrParams.hParams.maxAmberHue )
    {
        cerr << "  In spot lights recognition by hue parameters." << endl
             << "  Min red hue can not be lower or equal than max amber hue: "
             << slrParams.hParams.minRedHue << " <= " << slrParams.hParams.maxAmberHue
             << endl << endl;
        return 1;
    }

    // - Min amber hue

    if( slrParams.hParams.minAmberHue < 0 || slrParams.hParams.minAmberHue > 180 )
    {
        cerr << "  In spot lights recognition by hue parameters." << endl
             << "  Invalid min amber hue: " << slrParams.hParams.minAmberHue << endl
             << endl;
        return 1;
    }

    // - Max amber hue

    if( slrParams.hParams.maxAmberHue < 0 || slrParams.hParams.maxAmberHue > 180 )
    {
        cerr << "  In spot lights recognition by hue parameters." << endl
             << "  Invalid max amber hue: " << slrParams.hParams.maxAmberHue << endl
```

```
                << endl;
            return 1;
        }

        // - Min-Max amber hue

        if( slrParams.hParams.minAmberHue > slrParams.hParams.maxAmberHue )
        {
            cerr << "  In spot lights recognition by hue parameters." << endl
                 << "  Min amber hue can not be greater than max amber hue: "
                 << slrParams.hParams.minAmberHue << " > " << slrParams.hParams.maxAmberHue
                 << endl << endl;
            return 1;
        }

        // - Min amber hue and max green hue

        if( slrParams.hParams.minAmberHue <= slrParams.hParams.maxGreenHue )
        {
            cerr << "  In spot lights recognition by hue parameters." << endl
                 << "  Min amber hue can not be lower or equal than max green hue: "
                 << slrParams.hParams.minAmberHue << " <= " << slrParams.hParams.maxGreenHue
                 << endl << endl;
            return 1;
        }

        // - Min green hue

        if( slrParams.hParams.minGreenHue < 0 || slrParams.hParams.minGreenHue > 180 )
        {
            cerr << "  In spot lights recognition by hue parameters." << endl
                 << "  Invalid min green hue: " << slrParams.hParams.minGreenHue << endl
                 << endl;
            return 1;
        }

        // - Max green hue

        if( slrParams.hParams.maxGreenHue < 0 || slrParams.hParams.maxGreenHue > 180 )
        {
            cerr << "  In spot lights recognition by hue parameters." << endl
                 << "  Invalid max green hue: " << slrParams.hParams.maxGreenHue << endl
                 << endl;
            return 1;
        }

        // - Min-Max green hue

        if( slrParams.hParams.minGreenHue > slrParams.hParams.maxGreenHue )
        {
            cerr << "  In spot lights recognition by hue parameters." << endl
                 << "  Min green hue can not be greater than max green hue: "
                 << slrParams.hParams.minGreenHue << " > " << slrParams.hParams.maxGreenHue
                 << endl << endl;
            return 1;
        }

    }

    // ------------------------------------------------------------------2
    // Detection and recognition display

    // - Red recognition image

    if( drdParams.imageFileNames[0] == "null" )
    {
        cerr << "  In detection and recognition display parameters." << endl
             << "  Invalid red recognition image: " << drdParams.imageFileNames[0]
             << endl << endl;
        return 1;
    }

    // - Amber-up recognition image
```

```
if( drdParams.imageFileNames[1] == "null" )
{
   cerr << "  In detection and recognition display parameters." << endl
        << "  Invalid amber-up recognition image: " << drdParams.imageFileNames[1]
        << endl << endl;
   return 1;
}

// - Green recognition image

if( drdParams.imageFileNames[2] == "null" )
{
   cerr << "  In detection and recognition display parameters." << endl
        << "  Invalid green recognition image: " << drdParams.imageFileNames[2] << endl;
        << endl;
   return 1;
}

// - Amber-down recognition image

if( drdParams.imageFileNames[3] == "null" )
{
   cerr << "  In detection and recognition display parameters." << endl
        << "  Invalid amber-down recognition image: " << drdParams.imageFileNames[3]
        << endl << endl;
   return 1;
}

// - Amber-or-green recognition image

if( drdParams.imageFileNames[4] == "null" )
{
   cerr << "  In detection and recognition display parameters." << endl
        << "  Invalid amber-or-green recognition image: " << drdParams.imageFileNames[4]
        << endl << endl;
   return 1;
}

// - Amber-up-or-down recognition image

if( drdParams.imageFileNames[5] == "null" )
{
   cerr << "  In detection and recognition display parameters." << endl
        << "  Invalid amber-up-or-down recognition image: "<< drdParams.imageFileNames[5]
        << endl << endl;
   return 1;
}

// - Unknown recognition image

if( drdParams.imageFileNames[6] == "null" )
{
   cerr << "  In detection and recognition display parameters." << endl
        << "  Invalid unknown recognition image: " << drdParams.imageFileNames[6] << endl
        << endl
   return 1;
}

// - Detection draw color

if( ( drdParams.drawColor != "red" )  && ( drdParams.drawColor != "green"  ) &&
    ( drdParams.drawColor != "blue" ) && ( drdParams.drawColor != "yellow" ) )
{
   cerr << "  In detection and recognition display parameters." << endl
        << "  Invalid draw color: " << drdParams.drawColor << endl
        << endl;
   return 1;
}

// - Detection draw thickness

if( drdParams.drawThickness < 1 )
```

```
    {
        cerr << "  In detection and recognition display parameters." << endl
             << "  Invalid draw thickness: " << drdParams.drawThickness << endl
             << endl;
        Return 1;
    }

    // - Display scale factor

    if( drdParams.scaleFactor <= 0.0f )
    {
        cerr << "  In detection and recognition display parameters." << endl
             << "  Invalid scale factor: " <<  drdParams.scaleFactor << endl
             << endl;
        return 1;
    }

    return 0;

}

int loadFilesAndDevices( siParameters &siParams, tldParameters &tldParams,
                         sldParameters &sldParams, slrParameters &slrParams,
                         drdParameters &drdParams )
{
    // ---------------------------------------------------------------------------1
    // Load program files and devices

    // ---------------------------------------------------------------------2
    // Source input files and devices

    // - Load images list file

    if( siParams.listInput )
    {
        siParams.listFile.open( siParams.listFileName.c_str() );

        if( !siParams.listFile.is_open() )
        {
            cerr << "  Unable to load images list file: " << siParams.listFileName << endl;
            return 1;
        }
    }

    // - Load video file

    if( siParams.videoInput )
    {
        siParams.videoCapture.open( siParams.videoFileName );

        if ( !siParams.videoCapture.isOpened() )
        {
            cerr << "  Unable to load video file: " << siParams.videoFileName << endl
                 << endl;
            return 1;
        }
    }

    // - Load camera

    if( siParams.cameraInput )
    {
        siParams.videoCapture.open( siParams.cameraNumber );

        if ( !siParams.videoCapture.isOpened() )
        {
            cerr << "  Unable to load video camera number: " << siParams.cameraNumber << endl
                 << endl;
            return 1;
        }
    }

    // ---------------------------------------------------------------------2
```

```
// Traffic lights detection files

// - Load cascade classifier file

tldParams.cascade.load( tldParams.cascadeFileName );

if ( tldParams.cascade.empty() )
{
   cerr << "  Unable to load cascade classifier file: " << tldParams.cascadeFileName
        << endl << endl;
   return 1;
}

// ------------------------------------------------------------------2
// Spot lights recognition by matching

slrParams.mParams.matchingTemplates = vector<Mat> ( 3, Mat() );

// - Red matching template

slrParams.mParams.matchingTemplates[0] = imread( slrParams.mParams.templateFileNames[0]);
if( slrParams.mParams.matchingTemplates[0].empty() )
{
   cerr << "  Unable to load red matching template: "
        << slrParams.mParams.templateFileNames[0] << endl
        << endl;
   return 1;
 }

// - Amber matching template

slrParams.mParams.matchingTemplates[1] = imread( slrParams.mParams.templateFileNames[1]);
if( slrParams.mParams.matchingTemplates[1].empty() )
{
   cerr << "  Unable to load amber matching template: "
       << slrParams.mParams.templateFileNames[1] << endl
       << endl;
   return 1;
}

// - Green matching template

slrParams.mParams.matchingTemplates[2] = imread( slrParams.mParams.templateFileNames[2]);
if( slrParams.mParams.matchingTemplates[2].empty() )
{
   cerr << "  Unable to load green matching template: "
        << slrParams.mParams.templateFileNames[2] << endl
        << endl;
   return 1;
}

// ------------------------------------------------------------------2
// Detection and recognition display files

drdParams.recognitionImages = vector<Mat> ( 7, Mat() );

// - Load red recognition image

drdParams.recognitionImages[0] = imread( drdParams.imageFileNames[0] );
if( drdParams.recognitionImages[0].empty() )
{
   cerr << "  Unable to load red recognition image: " << drdParams.imageFileNames[0]
        << endl << endl;
   return 1;
}

// - Load amber-up recognition image

drdParams.recognitionImages[1] = imread( drdParams.imageFileNames[1] );
if( drdParams.recognitionImages[1].empty() )
{
   cerr << "  Unable to load amber-up recognition image: " << drdParams.imageFileNames[1]
        << endl << endl;
```

```
        return 1;
    }

    // - Load green recognition image

    drdParams.recognitionImages[2] = imread( drdParams.imageFileNames[2] );
    if( drdParams.recognitionImages[2].empty() )
    {
        cerr << "  Unable to load green recognition image: " << drdParams.imageFileNames[2]
             << endl << endl;
        return 1;
    }

    // - Load amber-down recognition image

    drdParams.recognitionImages[3] = imread( drdParams.imageFileNames[3] );
    if( drdParams.recognitionImages[3].empty() )
    {
        cerr << "  Unable to load amber-down recognition image: "
             << drdParams.imageFileNames[3] << endl << endl;
        return 1;
    }

    // - Load amber-or-green recognition image

    drdParams.recognitionImages[4] = imread( drdParams.imageFileNames[4] );
    if( drdParams.recognitionImages[4].empty() )
    {
        cerr << "  Unable to load amber-or-green recognition image: "
             << drdParams.imageFileNames[4] << endl << endl;
        return 1;
    }

    // - Load amber-up-or-down recognition image

    drdParams.recognitionImages[5] = imread( drdParams.imageFileNames[5] );
    if( drdParams.recognitionImages[5].empty() )
    {
        cerr << "  Unable to load amber-up-or-down recognition image: "
             << drdParams.imageFileNames[5] << endl << endl;
        return 1;
    }

    // - Load unknown recognition image

    drdParams.recognitionImages[6] = imread( drdParams.imageFileNames[6] );
    if( drdParams.recognitionImages[6].empty() )
    {
        cerr << "  Unable to load unknown recognition image: " << drdParams.imageFileNames[6]
             << endl << endl;
        return 1;
    }

    return 0;
}

void trafficLightsDetection( Mat &inputImage, vector<Rect> &boundingBoxes,
                             tldParameters tldParams )
{
    // ---------------------------------------------------------------------------1
    // Detect traffic lights bounding boxes using cascade classifier

    // ---------------------------------------------------------------------------2
    // Preprocess input image to improve cascade classifier detection

    // - Convert input image to greyscale

    Mat greyImage;
    cvtColor( inputImage, greyImage, CV_RGB2GRAY );

    // - Shrink image to increase processing speed

    Mat shrunkImage;
```

```
int shrunkWidth = cvRound( greyImage.cols * tldParams.shrinkFactor );
int shrunkHeight = cvRound( greyImage.rows * tldParams.shrinkFactor );
Size shrunkSize = Size( shrunkWidth, shrunkHeight );

resize( greyImage, shrunkImage, shrunkSize );

// - Crop image to reduce search area

int croppedHeight = cvRound( shrunkImage.rows * tldParams.cropFactor );
Rect croppedArea = Rect( 0, 0, shrunkImage.cols, croppedHeight );

Mat croppedImage = shrunkImage( croppedArea ).clone();

// - Blur image with gaussian to smooth

Mat blurredImage;
Size kernelSize = Size( tldParams.blurSize, tldParams.blurSize );
GaussianBlur( croppedImage, blurredImage, kernelSize, 0.0 );

// -----------------------------------------------------------------2
// Run cascade classifier and get detection bounding boxes

tldParams.cascade.detectMultiScale( blurredImage, boundingBoxes, tldParams.scaleFactor,
                                    0, 0, tldParams.minSize, tldParams.maxSize );

// -----------------------------------------------------------------2
// Enlarge bounding boxes if input image was shrunk before detection

// - Enlarge bounding boxes

for (int i = 0; i < int( boundingBoxes.size() ); i++ )
{
   boundingBoxes[i].x = cvRound( boundingBoxes[i].x / tldParams.shrinkFactor );
   boundingBoxes[i].y = cvRound( boundingBoxes[i].y / tldParams.shrinkFactor );
   boundingBoxes[i].width = cvRound( boundingBoxes[i].width / tldParams.shrinkFactor );
   boundingBoxes[i].height = cvRound( boundingBoxes[i].height / tldParams.shrinkFactor );
}

// - Check if bounding boxes are within limits

for ( int i = 0; i < int( boundingBoxes.size() ); i++ )
{
   if ( boundingBoxes[i].x < 0 )
   {
      boundingBoxes[i].x = 0;
   }
   if ( boundingBoxes[i].y < 0 )
   {
      boundingBoxes[i].y = 0;
   }
   if ( boundingBoxes[i].x + boundingBoxes[i].width > inputImage.cols )
   {
      boundingBoxes[i].x = inputImage.cols - boundingBoxes[i].width;
   }
   if ( boundingBoxes[i].y + boundingBoxes[i].height > inputImage.rows )
   {
      boundingBoxes[i].y = inputImage.rows - boundingBoxes[i].height;
   }
}

// -----------------------------------------------------------------2
// Group, sort and filter bounding boxes

// - Group bounding boxes by clustering and get their scores

vector<int> bboxesScores;
groupRectangles( boundingBoxes, bboxesScores, tldParams.minGroup );

// - Sort bounding boxes in descending order of score

for( int i = 0; i < int( boundingBoxes.size() ) - 1; i++ )
{
   for( int j = i + 1; j < int( boundingBoxes.size() ); j++ )
```

```
        {
            if( bboxesScores[i] < bboxesScores[j] )
            {
                swap( boundingBoxes[i], boundingBoxes[j] );
                swap( bboxesScores[i], bboxesScores[j] );
            }
        }
    }

    // - Filter bounding boxes by non maximun supression

    for( int i = 0; i < int( boundingBoxes.size() ) - 1; i++ )
    {
        for( int j = i + 1; j < int( boundingBoxes.size() ); j++ )
        {
            int x1 = max( boundingBoxes[i].x, boundingBoxes[j].x );
            int y1 = max( boundingBoxes[i].y, boundingBoxes[j].y );
            int x2 = min( boundingBoxes[i].x + boundingBoxes[i].width - 1,
                          boundingBoxes[j].x + boundingBoxes[j].width - 1 );
            int y2 = min( boundingBoxes[i].y + boundingBoxes[i].height - 1,
                          boundingBoxes[j].y + boundingBoxes[j].height - 1 );

            int width = x2 - x1 + 1;
            int height = y2 - y1 + 1;

            if( width > 0 && height > 0 )
            {
                int interseccionArea = width * height;
                int unionArea = boundingBoxes[i].area() + boundingBoxes[j].area() -
interseccionArea;
                float overlap = float( interseccionArea ) / unionArea;

                if( overlap >= tldParams.minOverlap )
                {
                    boundingBoxes.erase( boundingBoxes.begin() + j );
                    bboxesScores.erase( bboxesScores.begin() + j );
                    j--;
                }
            }
        }
    }

    // - Filter bounding boxes by max number of them

    for( int i = tldParams.maxNumber; i < int( boundingBoxes.size() ); i++ )
    {
        boundingBoxes.erase( boundingBoxes.begin() + i );
        bboxesScores.erase( bboxesScores.begin() + i );
    }

}

void blobsDetector( Mat &image, vector<Rect> &blobs, bdParameters bdParams )
{
    // ------------------------------------------------------------------------1
    // Detect blobs

    for ( int thresh = bdParams.minThresh; thresh <= bdParams.maxThresh; thresh +=
          bdParams.threshStep )
    {
        // ------------------------------------------------------------2
        // Binarizes greyscale image using thresholding step

        Mat binarizedImage;
        threshold( image, binarizedImage, thresh, 255, THRESH_BINARY );

        // ------------------------------------------------------------2
        // Find blobs contours of binarized image

        vector < vector<Point> > contours;
        Mat tmpBinaryImage = binarizedImage.clone();
        findContours( tmpBinaryImage, contours, CV_RETR_LIST, CV_CHAIN_APPROX_NONE );
```

```
        // ------------------------------------------------------------2
        // Get final blobs

        for ( size_t contourIdx = 0; contourIdx < contours.size(); contourIdx++ )
        {
            // ------------------------------------------------3
            // - Get moments of each blob contour

            Moments moms = moments( Mat( contours[contourIdx] ) );

            // ------------------------------------------------3
            // Filter blob contours by area, circularity, inertia, and convexity

            // - Filter by area

            if ( bdParams.filterDiam )
            {
                double area = moms.m00;
                double minArea = CV_PI * bdParams.minDiam * bdParams.minDiam / 4.0;
                double maxArea = CV_PI * bdParams.maxDiam * bdParams.maxDiam / 4.0;
                if ( area < minArea || area > maxArea )
                    continue;
            }

            // - Filter by circularity

            if ( bdParams.filterCirc )
            {
                double area = moms.m00;
                double perimeter = arcLength( Mat( contours[contourIdx] ), true );
                double ratio = 4 * CV_PI * area / ( perimeter * perimeter );
                if ( ratio < bdParams.minCirc || ratio > bdParams.maxCirc )
                    continue;
            }

            // - Filter by inertia ratio

            if ( bdParams.filterInert )
            {
                double denominator = sqrt( pow( 2 * moms.mu11, 2 ) +
                                           pow( moms.mu20 - moms.mu02, 2 ) );
                const double eps = 1e-2;
                double ratio;
                if ( denominator > eps )
                {
                    double cosmin = ( moms.mu20 - moms.mu02 ) / denominator;
                    double sinmin = 2 * moms.mu11 / denominator;
                    double cosmax = -cosmin;
                    double sinmax = -sinmin;

                    double imin = 0.5 * ( moms.mu20 + moms.mu02 ) - 0.5 *
                                  ( moms.mu20 - moms.mu02 ) * cosmin - moms.mu11 * sinmin;
                    double imax = 0.5 * ( moms.mu20 + moms.mu02 ) - 0.5 *
                                  ( moms.mu20 - moms.mu02 ) * cosmax - moms.mu11 * sinmax;
                    ratio = imin / imax;
                }
                else
                {
                    ratio = 1;
                }

                if ( ratio < bdParams.minInert || ratio > bdParams.maxInert )
                    continue;
            }

            // - Filter by convexity

            if ( bdParams.filterConvex )
            {
                vector < Point > hull;
                convexHull( Mat( contours[contourIdx] ), hull );
                double area = contourArea( Mat( contours[contourIdx] ) );
                double hullArea = contourArea( Mat( hull ) );
```

```
            double ratio = area / hullArea;
            if ( ratio < bdParams.minConvex || ratio > bdParams.maxConvex )
               continue;
         }

         // ------------------------------------------------3
         // Calculate blobs

         // - Calculate blob location

         Point location;
         location.x = cvRound( moms.m10 / moms.m00 );
         location.y = cvRound(  moms.m01 / moms.m00 );

         // - Calculate blobs radius

         double radius;
         vector<double> dists;

         for ( size_t pointIdx = 0; pointIdx < contours[contourIdx].size(); pointIdx++ )
         {
            Point pt = contours[contourIdx][pointIdx];
            dists.push_back( norm( location - pt ) );
         }
         std::sort( dists.begin(), dists.end() );
         radius = ( dists[ ( dists.size() - 1 ) / 2 ] + dists[ dists.size() / 2 ] ) / 2.;

         // - Calculate blobs with location and radius

         Rect auxBlob;
         auxBlob.x = cvRound( location.x - radius + 1 );
         auxBlob.y = cvRound( location.y - radius + 1 );
         auxBlob.width = cvRound( 2.0 * radius );
         auxBlob.height = auxBlob.width;

         // - Check if blobs are within image

         if( auxBlob.x < 0 )
         {
            auxBlob.x = 0;
         }

         if( auxBlob.y < 0 )
         {
            auxBlob.y = 0;
         }

         if( auxBlob.x + auxBlob.width > image.cols )
         {
            auxBlob.width = image.cols - auxBlob.x;
            auxBlob.height = auxBlob.width;
         }

         if( auxBlob.y + auxBlob.height > image.rows )
         {
            auxBlob.height = image.rows - auxBlob.y;
            auxBlob.width = auxBlob.height;
         }

         // - Save blobs

         blobs.push_back( auxBlob );

      }

   }

}

void spotLightsDetection( Mat &inputImage, vector<Rect> &boundingBoxes,
                          vector<Rect> &boundingBlobs, sldParameters sldParams )
{
   // ------------------------------------------------------------------------1
   // Detect spot lights bounding blobs using blob detector
```

```
void spotLightsDetection( Mat &inputImage, vector<Rect> &boundingBoxes,
                          vector<Rect> &boundingBlobs, sldParameters sldParams )
{
   // ------------------------------------------------------------------------1
   // Detect spot lights bounding blobs using blob detector

   for( int i = 0; i < int( boundingBoxes.size() ); i++ )
   {
      // ---------------------------------------------------------------2
      // Get traffic light ROI delimited by bounding box

      Mat inputRoi = inputImage( boundingBoxes[i] ).clone();

      // ---------------------------------------------------------------2
      // Preprocess traffic light ROI

      // - Convert ROI to greyscale

      Mat greyRoi;
      cvtColor( inputRoi, greyRoi, CV_RGB2GRAY );

      // - Normalize ROI between min and max values

      Mat normalizedRoi;
      normalize( greyRoi, normalizedRoi, 0, 255, NORM_MINMAX );

      // - Filter ROI to remove noise

      Mat filteredRoi;
      Size kernelSize = Size( sldParams.blurSize, sldParams.blurSize );
      GaussianBlur( normalizedRoi, filteredRoi, kernelSize, 0.0 );

      // ---------------------------------------------------------------2
      // Run blobs detector and get detected blobs

      vector<Rect> detectedBlobs;
      blobsDetector( filteredRoi, detectedBlobs, sldParams.bdParams );

      // ---------------------------------------------------------------2
      // Group, sort and filter bounding blobs

      // - Group detected blobs by clustering and get their scores

      vector<int> blobsScores;
      groupRectangles( detectedBlobs, blobsScores, sldParams.minGroup );

      // - Sort detected blobs in descending order of score

      for( int j = 0; j < int( detectedBlobs.size() ) - 1; j++ )
      {
         for( int k = j + 1; k < int( detectedBlobs.size() ); k++ )
         {
            if( blobsScores[j] < blobsScores[k] )
            {
               swap( detectedBlobs[j], detectedBlobs[k] );
               swap( blobsScores[j], blobsScores[k] );
            }
         }
      }

      // - Filter detected blobs by non maximun supression

      for( int j = 0; j < int( detectedBlobs.size() ) - 1; j++ )
      {
         for( int k = j + 1; k < int( detectedBlobs.size() ); k++ )
         {
            int x1 = max( detectedBlobs[j].x, detectedBlobs[k].x );
            int y1 = max( detectedBlobs[j].y, detectedBlobs[k].y );
            int x2 = min( detectedBlobs[j].x + detectedBlobs[j].width - 1,
                          detectedBlobs[k].x + detectedBlobs[k].width - 1 );
            int y2 = min( detectedBlobs[j].y + detectedBlobs[j].height - 1,
                          detectedBlobs[k].y + detectedBlobs[k].height - 1 );
```

```
            int width = x2 - x1 + 1;
            int height = y2 - y1 + 1;

            if( width > 0 && height > 0 )
            {
               int interseccionArea = width * height;
               int unionArea = detectedBlobs[j].area() + detectedBlobs[k].area() -
                           interseccionArea;
               float overlap = float( interseccionArea ) / unionArea;

               if( overlap >= sldParams.minOverlap )
               {
                  detectedBlobs.erase( detectedBlobs.begin() + k );
                  blobsScores.erase( blobsScores.begin() + k );
                  k--;
               }
            }
         }
      }

      // ------------------------------------------------------------------2
      // Update bounding blobs and bounding boxes

      // - Save bounding blob with more score

      if( detectedBlobs.size() > 0 )
      {
         boundingBlobs.push_back( detectedBlobs[0] );
      }

      // - Remove bounding boxes that not have bounding blobs

      if( detectedBlobs.size() == 0 )
      {
         boundingBoxes.erase( boundingBoxes.begin() + i );
         i--;
      }

   }

}

void spotLightsRecognitionByPosition( Mat &inputImage, vector<Rect> &boundingBoxes,
                                      vector<Rect> &boundingBlobs,
                                      vector<slColor> &spotLightColors, pParameters pParams)
{
   // ----------------------------------------------------------------------------1
   // Recognize spot lights color by position

   for( int i = 0; i < int( boundingBoxes.size() ); i++ )
   {
      // ------------------------------------------------------------------2
      // Get traffic light ROI delimited by bounding box

      Mat inputRoi = inputImage( boundingBoxes[i] ).clone();

      // ------------------------------------------------------------------2
      // Preprocess traffic light ROI

      // - Convert ROI to greyscale

      Mat greyRoi;
      cvtColor( inputRoi, greyRoi, CV_RGB2GRAY );

      // - Normalize ROI between min and max values

      Mat normalizedRoi;
      normalize( greyRoi, normalizedRoi, 0, 255, NORM_MINMAX );

      // - Filter ROI to remove unsharp edges

      Mat blurredRoi;
      Size kernelSize = Size( pParams.blurSize, pParams.blurSize );
```

```
        GaussianBlur( normalizedRoi, blurredRoi, kernelSize, 0.0 );

        // ------------------------------------------------------------2
        // Get horizontal edges of ROI

        // - Apply sobel operator to ROI to get horizontal edges

        Mat sobelRoi;
        Sobel( blurredRoi, sobelRoi, CV_16SC1, 0, 1, pParams.sobelSize );
        convertScaleAbs( sobelRoi, sobelRoi );

        // - Keep sharp edges by thresholding

        Mat edgesRoi;
        threshold( sobelRoi, edgesRoi, pParams.edgesThresh, 1, THRESH_BINARY );

        // ------------------------------------------------------------2
        // Calculate upper and lower lateral histograms of edges ROI

        // - Calculate spot light bounding lines

        int boundInc = cvRound( pParams.boundFactor * boundingBlobs[i].width );

        int upperBoundLine = boundingBlobs[i].y - boundInc;
        int lowerBoundLine = boundingBlobs[i].y + boundingBlobs[i].height - 1 + boundInc;
        int leftBoundLine  = boundingBlobs[i].x - boundInc;
        int rightBoundLine = boundingBlobs[i].x + boundingBlobs[i].width - 1 + boundInc;

        // - Check if spot light bounding lines are within limits

        if( upperBoundLine < 0 )
        {
           upperBoundLine = 0;
        }

        if( lowerBoundLine >= inputRoi.rows )
        {
           lowerBoundLine = inputRoi.rows - 1 ;
        }

        if( leftBoundLine < 0 )
        {
           leftBoundLine = 0;
        }

        if( rightBoundLine >= inputRoi.cols )
        {
           rightBoundLine = inputRoi.cols - 1;
        }
        // - Calculate upper and lower lateral histograms limited by bounding lines

        vector<int> upperHist( inputRoi.rows, 0 );
        vector<int> lowerHist( inputRoi.rows, 0 );

        for( int j = 0; j < inputRoi.rows; j++ )
        {
           if( j <= upperBoundLine )
           {
              for( int k = 0; k < inputRoi.cols; k++ )
              {
                 if( k > leftBoundLine && k < rightBoundLine )
                 {
                    upperHist[j] += edgesRoi.at<uchar>(j,k);
                 }
              }
           }

           if( j >= lowerBoundLine )
           {
              for( int k = 0; k < inputRoi.cols; k++ )
              {
                 if( k > leftBoundLine && k < rightBoundLine )
                 {
```

```
                lowerHist[j] += edgesRoi.at<uchar>(j,k);
            }
        }
    }
}

// ------------------------------------------------------------2
// Find upper and lower edges of traffic light unit

// - Filter edges in lateral histograms with enough width

int minWidth = cvRound( pParams.widthFactor * ( rightBoundLine - leftBoundLine + 1 ));

for( int j = 0; j < int( upperHist.size() ); j++ )
{
    if( upperHist[j] >= minWidth )
    {
        upperHist[j] = 1;
    }
    else
    {
        upperHist[j] = 0;
    }
}

for( int j = 0; j < int( lowerHist.size() ); j++ )
{
    if( lowerHist[j] >= minWidth )
    {
        lowerHist[j] = 1;
    }
    else
    {
        lowerHist[j] = 0;
    }
}

// - Find upper and lower edges by max value of lateral histograms

int upperEdge = 0;
bool upperFound = false;

for( int j = int( upperHist.size() ) - 1; j >= 0 ; j-- )
{
    if( upperHist[j] == 1 )
    {
        upperEdge = j;
        upperFound = true;
        break;
    }
}

int lowerEdge = lowerHist.size() - 1;
bool lowerFound = false;

for( int j = 0; j < int( lowerHist.size() ); j++ )
{
    if( lowerHist[j] == 1 )
    {
        lowerEdge = j;
        lowerFound = true;
        break;
    }
}

// ------------------------------------------------------------2
// Identify spot lights color by position

int distEdges = lowerEdge - upperEdge + 1;
int maxDist = cvRound( pParams.distFactor * boundingBlobs[i].width );
int blobCenterHeight =  ( boundingBlobs[i].y + cvRound( boundingBlobs[i].width /
                                                        2.0f ) - 1 ) - upperEdge ;
```

```
        slColor spotLightColor;

        if( distEdges < maxDist )
        {
            if( upperFound && !lowerFound )
            {
                spotLightColor = red;
            }
            else if( !upperFound && lowerFound )
            {
                spotLightColor = amberorgreen;
            }
            else
            {
                spotLightColor = unknown;
            }
        }
        else if( distEdges < ( 2 * maxDist ) )
        {
            if( upperFound && !lowerFound )
            {
                if( blobCenterHeight < cvRound( distEdges / 2.0f ) )
                {
                    spotLightColor = red;
                }
                }
                else
                {
                    spotLightColor = amberup;
                }
            }
            else if( !upperFound && lowerFound )
            {
                if( blobCenterHeight < cvRound( distEdges / 2.0f ) )
                {
                    spotLightColor = amberup;
                }
                else
                {
                    spotLightColor = amberorgreen;
                }
            }
            else
            {
                spotLightColor = unknown;
            }
        }
        else
        {
            if( blobCenterHeight < cvRound( distEdges / 3.0f ) )
            {
                spotLightColor = red;
            }
            else if ( blobCenterHeight < cvRound( 2.0f * distEdges / 3.0f ) )
            {
                spotLightColor = amberup;
            }
            else
            {
                spotLightColor = amberorgreen;
            }
        }

        spotLightColors.push_back( spotLightColor );

    }

}

void spotLightsRecognitionByMatching( Mat &inputImage, vector<Rect> &boundingBoxes,
                                      vector<Rect> &boundingBlobs,
                                      vector<slColor> &spotLightColors, mParameters mParams)
{
```

```
// ------------------------------------------------------------------------1
// Recognize spot lights color by matching templates

for( int i = 0; i < int( boundingBlobs.size() ); i++ )
{
   // ---------------------------------------------------------------2
   // Adapt template images to input image

   // - Convert template images to greyscale

   vector<Mat> greyTemplates( mParams.matchingTemplates.size() );

   for( int j = 0; j < int( mParams.matchingTemplates.size() ); j++ )
   {
      cvtColor( mParams.matchingTemplates[j], greyTemplates[j], CV_RGB2GRAY );
   }

            // - Resize template images and lights box to match with spot light size

   vector<Mat> resizedTemplates( mParams.matchingTemplates.size() );
   vector<Rect> resizedLightBoxes( mParams.lightBoxes.size() );

   for( int j = 0; j < int( mParams.matchingTemplates.size() ); j++ )
   {
      float resizeFactor = float( boundingBlobs[i].width ) /
                           mParams.matchingTemplates[j].cols;

      // - Resize templates

      int resizedWidth = cvRound( mParams.matchingTemplates[j].cols * resizeFactor );
      int resizedHeight = cvRound( mParams.matchingTemplates[j].rows * resizeFactor );
      Size resizedSize = Size( resizedWidth, resizedHeight );

      resize( greyTemplates[j], resizedTemplates[j], resizedSize );

      // - Resize lights box

      resizedLightBoxes[j].x = cvRound( resizeFactor * mParams.lightBoxes[j].x );
      resizedLightBoxes[j].y = cvRound( resizeFactor * mParams.lightBoxes[j].y );
      resizedLightBoxes[j].width = cvRound( resizeFactor * mParams.lightBoxes[j].width );
      resizedLightBoxes[j].height=cvRound( resizeFactor * mParams.lightBoxes[j].height );
   }

   // ---------------------------------------------------------------2
   // Obtain matching scores for each template

   vector<float> matchingScores( mParams.matchingTemplates.size(), 0.0f );

   for( int j = 0; j < int( mParams.matchingTemplates.size() ); j++ )
   {
      // - Obtain ROI where do matching with templates

      Rect matchingBox;
      matchingBox.x = boundingBoxes[i].x + boundingBlobs[i].x - resizedLightBoxes[j].x;
      matchingBox.y =  boundingBoxes[i].y + boundingBlobs[i].y - resizedLightBoxes[j].y;
      matchingBox.width = resizedTemplates[j].cols;
      matchingBox.height = resizedTemplates[j].rows;

      if( matchingBox.x < 0 || matchingBox.y < 0 || matchingBox.width <= 0 ||
          matchingBox.height <= 0 )
         continue;

      Mat matchingRoi = inputImage( matchingBox ).clone();

      // - Convert ROI to greyscale

      Mat greyRoi;
      cvtColor( matchingRoi, greyRoi, CV_RGB2GRAY );

      // - Normalize ROI between min and max values

      Mat normalizedRoi;
      normalize( greyRoi, normalizedRoi, 0, 255, NORM_MINMAX );
```

```
            // - Filter ROI to smooth

            Mat bluredRoi;
            Size kernelSize = Size( mParams.blurSize,  mParams.blurSize );
            GaussianBlur( normalizedRoi, bluredRoi, kernelSize, 0.0 );

            // - Do matching template

            Mat matchingMatrix = Mat( 1, 1, CV_32FC1 );
            matchTemplate(bluredRoi, resizedTemplates[j], matchingMatrix, CV_TM_CCORR_NORMED );

            // - Get matching score

            matchingScores[j] = matchingMatrix.at<float>(0,0);

        }

        // ------------------------------------------------------------2
        // Identify spot light color with matching score

        // - Find matching with max score

        float maxScore = 0.0f;
        int maxIndex = -1;

        for( int j = 0; j < int( mParams.matchingTemplates.size() ); j++ )
        {
            if( matchingScores[j] > maxScore )
            {
                maxScore = matchingScores[j];
                maxIndex = j;
            }
        }

        // - Identify spot light color with max score

        slColor spotLightColor;

        if( maxIndex == 0 )
        {
            spotLightColor = red;
        }
        else if( maxIndex == 1 )
        {
            spotLightColor = amberup;
        }
        else if( maxIndex == 2 )
        {
            spotLightColor = amberorgreen;
        }
        else
        {
            spotLightColor = unknown;
        }

        spotLightColors.push_back( spotLightColor );
    }

}

void spotLightsRecognitionByHue( Mat &inputImage, vector<Rect> &boundingBoxes,
                                 vector<Rect> &boundingBlobs,
                                 vector<slColor> &spotLightColors, hParameters hParams )
{
    // ------------------------------------------------------------------------1
    // Recognize spot lights color by hue

    for( int i = 0; i < int( boundingBoxes.size() ); i++ )
    {
        // ------------------------------------------------------------2
        // Get traffic light ROI
```

```
        Mat inputRoi = inputImage( boundingBoxes[i] ).clone();

        // ------------------------------------------------------------2
        // Build spot light mask

        // - Get bounding blob circle

        int blobRadius = cvRound( boundingBlobs[i].width / 2.0f );
        Point blobCenter;
        blobCenter.x = boundingBlobs[i].x + blobRadius - 1;
        blobCenter.y = boundingBlobs[i].y + blobRadius - 1;

        // - Draw bounding blob circle in spot light mask considering light halo

        Mat lightMask = Mat( inputRoi.rows, inputRoi.cols, CV_8UC1, Scalar( 0 ) );

        if( hParams.intHaloFactor == 0.0f && hParams.extHaloFactor == 0.0f )
        {
           circle( lightMask, blobCenter, blobRadius, Scalar( 255 ), -1 );
        }
        else
        {
           int intHaloRadius = cvRound( hParams.intHaloFactor * blobRadius );
           int extHaloRadius = cvRound( hParams.extHaloFactor * blobRadius );
           circle( lightMask, blobCenter, extHaloRadius, Scalar( 255 ), -1 );
           circle( lightMask, blobCenter, intHaloRadius, Scalar( 0 ), -1 );
        }

        // ------------------------------------------------------------2
        // Preprocess traffic light ROI

        // - Convert ROI to HSL space color

        Mat hlsRoi;
        cvtColor( inputRoi, hlsRoi, CV_RGB2HLS );

        // - Blur ROI to smooth

        Mat filteredRoi;
        Size kernelSize( hParams.blurSize, hParams.blurSize );
        GaussianBlur( hlsRoi, filteredRoi, kernelSize, 0.0 );

        // ------------------------------------------------------------2
        // Get mean hue

        // - Calculate hue histogram

        int hueChannel = 0;
        Mat hueHistogram;
        int histSize = 180;
        float hueRanges[] = { 0.0, 180.0f };
        const float* hueRange[] = { hueRanges };

        calcHist( &filteredRoi, 1, &hueChannel, lightMask, hueHistogram, 1, &histSize,
                  hueRange );

        // - Ignore histogram values below max histogram value rate

        double maxValue;
        minMaxLoc( hueHistogram, 0, &maxValue, 0, 0 );

        int threshValue = cvRound( hParams.histFactor * maxValue );

        for( int j = 0; j < hueHistogram.rows; j++ )
        {
           if( hueHistogram.at<float>(j) < threshValue )
           {
              hueHistogram.at<float>(j) = 0;
           }
        }

        // - Get mean hue
```

```
        float sumHue = 0.0f;
        float numHue = 0.0f;

        for( int j = 0; j < hueHistogram.rows; j++ )
        {
            sumHue += hueHistogram.at<float>(j) * j;
            numHue += hueHistogram.at<float>(j);
        }

        int meanHue = cvRound( sumHue / numHue );

        // ------------------------------------------------------------2
        // Identify light color by mean hue

        slColor spotLightColor;

        if( meanHue >= hParams.minRedHue && meanHue <= hParams.maxRedHue )
        {
            spotLightColor = red;
        }
        else if( meanHue >= hParams.minAmberHue && meanHue <= hParams.maxAmberHue )
        {
            spotLightColor = amberupordown;
        }
        else if( meanHue >= hParams.minGreenHue && meanHue <= hParams.maxGreenHue )
        {
            spotLightColor = green;
        }
        else
        {
            spotLightColor = unknown;
        }

        spotLightColors.push_back ( spotLightColor );

    }
}

void spotLigthsRecognition( Mat &inputImage, vector<Rect> &boundingBoxes,
                            vector<Rect> &boundingBlobs, vector<slColor> &spotLightColors,
                            slrParameters slrParams )
{
    // ---------------------------------------------------------------------------1
    // Recognize spot lights color by position, matching or hue

    if( slrParams.recognType == "pos" )
    {
        // ------------------------------------------------------------2
        // Recognize spot lights color by position

        spotLightsRecognitionByPosition( inputImage, boundingBoxes, boundingBlobs,
                                         spotLightColors, slrParams.pParams );

    }
    else if( slrParams.recognType == "match" )
    {
        // ------------------------------------------------------------2
        // Recognize spot lights color by matching

        spotLightsRecognitionByMatching( inputImage, boundingBoxes, boundingBlobs,
                                         spotLightColors, slrParams.mParams );

    }
    else if( slrParams.recognType == "hue" )
    {
        // ------------------------------------------------------------2
        // Recognize spot lights color by hue

        spotLightsRecognitionByHue( inputImage, boundingBoxes, boundingBlobs, spotLightColors,
                                    slrParams.hParams );

    }
    else if( slrParams.recognType == "pos-hue" )
```

```
    {
        // ------------------------------------------------------------2
        // Recognize spot lights color by position and hue

        // - Recognize spot lights color by position

        vector<slColor> posSpotLightColors;

        spotLightsRecognitionByPosition( inputImage, boundingBoxes, boundingBlobs,
                                         posSpotLightColors, slrParams.pParams );

        // - Recognize spot lights color by hue

        vector<slColor> hueSpotLightColors;

        spotLightsRecognitionByHue( inputImage, boundingBoxes, boundingBlobs,
                                    hueSpotLightColors, slrParams.hParams );

        // - Select final spot lights color by combination of position and hue recognition

        slColor spotLightColor;

        for( int i = 0; i < int( boundingBlobs.size() ); i++ )
        {
            if( posSpotLightColors[i] == red && hueSpotLightColors[i] == red )
            {
                spotLightColor = red;
            }
            else if( posSpotLightColors[i] == amberup && hueSpotLightColors[i] ==
                     amberupordown )
            {
                spotLightColor = amberup;
            }
            else if( posSpotLightColors[i] == amberorgreen && hueSpotLightColors[i] ==
                     amberupordown )
            {
                spotLightColor = amberdown;
            }
            else if( posSpotLightColors[i] == amberorgreen && hueSpotLightColors[i] == green )
            {
                spotLightColor = green;
            }
            else
            {
                spotLightColor = unknown;
            }

            spotLightColors.push_back( spotLightColor );
        }

    }
    else if( slrParams.recognType == "match-hue" )
    {
        // ------------------------------------------------------------2
        // Recognize spot lights color by matching and hue

        // - Recognize spot lights color by matching

        vector<slColor> matchSpotLightsColor;

        spotLightsRecognitionByMatching( inputImage, boundingBoxes, boundingBlobs,
                                         matchSpotLightsColor, slrParams.mParams );

        // - Recognize spot lights color by hue

        vector<slColor> hueSpotLightsColor;

        spotLightsRecognitionByHue( inputImage, boundingBoxes, boundingBlobs,
                                    hueSpotLightsColor, slrParams.hParams );

        // - Select final spot lights color by combination of matching and hue recognition

        slColor spotLightColor;
```

```
      for( int i = 0; i < int( boundingBlobs.size() ); i++ )
      {
         if( matchSpotLightsColor[i] == red && hueSpotLightsColor[i] == red )
         {
            spotLightColor = red;
         }
         else if( matchSpotLightsColor[i] == amberup && hueSpotLightsColor[i] ==
                  amberupordown )
         {
            spotLightColor = amberup;
         }
         else if( matchSpotLightsColor[i] == amberorgreen && hueSpotLightsColor[i] ==
                  amberupordown )
         {
            spotLightColor = amberdown;
         }
         else if( matchSpotLightsColor[i] == amberorgreen && hueSpotLightsColor[i] == green)
         {
            spotLightColor = green;
         }
         else
         {
            spotLightColor = unknown;
         }

         spotLightColors.push_back( spotLightColor );
      }

   }

}

void detectionAndRecognitionDisplay( Mat &inputImage, vector<Rect> &boundingBoxes,
                                     vector<Rect> &boundingBlobs,
                                     vector<slColor> &spotLightColors,
                                     drdParameters drdParams )
{
   // ----------------------------------------------------------------------------1
   // Display traffic lights detection and recognition

   // ---------------------------------------------------------------------2
   // Sort detected traffic lights from image left to right

   for( int i = 0; i < int( boundingBoxes.size() ) - 1; i++ )
   {
      for( int j = i + 1; j < int( boundingBoxes.size() ); j++ )
      {
         if( boundingBoxes[i].x < boundingBoxes[j].x )
         {
            swap( boundingBoxes[i], boundingBoxes[j] );
            swap( boundingBlobs[i], boundingBlobs[j] );
            swap( spotLightColors[i], spotLightColors[j] );
         }
      }
   }

   // ---------------------------------------------------------------------2
   // Draw bounding boxes and bounding blobs in detection image

   // - Select draw color

   Scalar drawColor;

   if( drdParams.drawColor == "red" )
   {
      drawColor = Scalar( 0, 0, 255 );
   }
   else if( drdParams.drawColor == "green" )
   {
      drawColor = Scalar( 0, 255, 0 );
   }
   else if( drdParams.drawColor == "blue" )
```

```
    {
       drawColor = Scalar( 255, 0, 0 );
    }
    else if( drdParams.drawColor == "yellow" )
    {
       drawColor = Scalar( 0, 255, 255 );
    }

    // - Draw bounding boxes

    Mat detectionImage = inputImage.clone();

    for( int i = 0; i < int( boundingBoxes.size() ); i++ )
    {
       rectangle( detectionImage,
                  Point( boundingBoxes[i].x, boundingBoxes[i].y ),
                  Point( boundingBoxes[i].x + boundingBoxes[i].width, boundingBoxes[i].y +
                          boundingBoxes[i].height ),
                  drawColor, drdParams.drawThickness, CV_AA );
    }

    // - Draw bounding blobs
    for( int i = 0; i < int( boundingBlobs.size() ); i++ )
    {
       Mat detectionRoi = detectionImage( boundingBoxes[i] );

       int blobRadius = cvRound( boundingBlobs[i].width / 2.0f );
       Point blobCenter;
       blobCenter.x = boundingBlobs[i].x + blobRadius;
       blobCenter.y = boundingBlobs[i].y + blobRadius;

       circle( detectionRoi, blobCenter, blobRadius, drawColor, drdParams.drawThickness,
               CV_AA );
    }

    // ------------------------------------------------------------------2
    // Create displayed image with detection image and recognition images

    // - Calculate width and height of displayed image

    int displayedWidth;
    int recognImagesWidth = boundingBoxes.size() * drdParams.recognitionImages[0].cols;

    if( inputImage.cols > recognImagesWidth )
    {
       displayedWidth = inputImage.cols;
    }
    else
    {
       displayedWidth = recognImagesWidth;
    }

    int displayedHeight = inputImage.rows + drdParams.recognitionImages[0].rows;

    Size displayedSize( displayedWidth, displayedHeight );

    // - Create displayed image

    Mat displayedImage( displayedSize, detectionImage.type() );

    // - Add detection image to displayed image

    Rect detectionBox;
    detectionBox.x = displayedImage.cols - detectionImage.cols;
    detectionBox.y = 0;
    detectionBox.width = detectionImage.cols;
    detectionBox.height = detectionImage.rows;

    Mat detectionRoi = displayedImage( detectionBox );
    detectionImage.copyTo( detectionRoi );

    // - Add recognition images to displayed image
```

```
    for( int i = 0; i < int( boundingBoxes.size() ); i++ )
    {
        Rect recognitionBox;
        recognitionBox.x = displayedImage.cols - drdParams.recognitionImages[0].cols *(i + 1);
        recognitionBox.y = detectionImage.rows;
        recognitionBox.width = drdParams.recognitionImages[0].cols;
        recognitionBox.height = drdParams.recognitionImages[0].rows;

        Mat recognitionRoi = displayedImage( recognitionBox );

        switch( spotLightColors[i] )
        {
            case red:

                drdParams.recognitionImages[0].copyTo( recognitionRoi );
                break;

            case amberup:

                drdParams.recognitionImages[1].copyTo( recognitionRoi );
                break;

            case green:

                drdParams.recognitionImages[2].copyTo( recognitionRoi );
                break;

            case amberdown:

                drdParams.recognitionImages[3].copyTo( recognitionRoi );
                break;

            case amberorgreen:

                drdParams.recognitionImages[4].copyTo( recognitionRoi );
                break;

            case amberupordown:

                drdParams.recognitionImages[5].copyTo( recognitionRoi );
                break;

            case unknown:

                drdParams.recognitionImages[6].copyTo( recognitionRoi );
                break;
        }
    }

    // -----------------------------------------------------------------2
    // Scale displayed image

    Mat scaledImage;
    int scaledWidth = cvRound( displayedImage.cols * drdParams.scaleFactor );
    int scaledHeight = cvRound( displayedImage.rows * drdParams.scaleFactor );
    Size scaledSize = Size( scaledWidth, scaledHeight );

    resize( displayedImage, scaledImage, scaledSize );

    // -----------------------------------------------------------------2
    // Display detection and recognition image

    imshow( "Display traffic lights detection and recognition", scaledImage );

}

// ------------------------------------------------------------------------------------------0
// Main

int main( int argc, char** argv )
{
    // ---------------------------------------------------------------------------1
    // Print title
```

```
cout << "+--------------------------------------------------------------------------+"
     << endl
     << "|                  TRAFFIC LIGHTS DETECTION AND RECOGNITION                |"
     << endl
     << "|                           By Victor Alonso Mendieta                      |"
     << endl
     << "|                                 October 2013                             |"
     << endl
     << "|                                                                          |"
     << endl
     << "|                  Traffic lights detection and recognition                |"
     << endl
     << "|             using cascade classifier with HAAR-LBP features             |"
     << endl
     << "|                     Compiled with OpenCV version 2.4.6                   |"
     << endl
     << "+--------------------------------------------------------------------------+"
     << endl;

// ------------------------------------------------------------------------1
// Define and initialize variables for program parameters

// - Define parameters

siParameters siParams;
tldParameters tldParams;
sldParameters sldParams;
slrParameters slrParams;
drdParameters drdParams;
bool usageHelp = false;

// - Initialize parameters

initializeParameters( siParams, tldParams, sldParams, slrParams, drdParams,
                      usageHelp );

// ------------------------------------------------------------------------1
// Read and print program parameters

// - Read program parameters

int readResult = readParameters( argc, argv, siParams, tldParams, sldParams, slrParams,
                 drdParams, usageHelp );

if( readResult == 1 )
{
   return 1;
}

// - Print usage help

if( usageHelp )
{
   cout << "Usage help..." << endl
        << endl;

   return 0;
}
else
{

// - Print program parameters

cout << "Program parameters..." << endl
     << endl;

   printParameters( siParams, tldParams, sldParams, slrParams, drdParams );

}

// ------------------------------------------------------------------------1
// Check program parameters
```

```
cout << "+--------------------------------------------------------------------------------+"
     << endl
     << "Checking program parameters..." << endl
     << endl;

int checkResult = checkParameters( siParams, tldParams, sldParams, slrParams, drdParams);

if( checkResult == 1 )
{
   return 1;
}

cout << "  Done" << endl
     << endl;

// ------------------------------------------------------------------------------1
// Load program files and devices

cout << "+--------------------------------------------------------------------------------+"
     << endl
     << "Loading program files and devices..." << endl
     << endl;

int loadResult = loadFilesAndDevices( siParams, tldParams, sldParams, slrParams,
                                      drdParams );

if( loadResult == 1 )
{
   return 1;
}

cout << "  Done" << endl
     << endl;

// ------------------------------------------------------------------------------1
// Start traffic lights detection and recognition

cout << "+--------------------------------------------------------------------------------+"
     << endl
     << "Start detection and recognition (Press ESC to exit)..." << endl
     << endl;

// ---------------------------------------------------------------------2
// Traffic lights detection and recognition from images list

if( siParams.listInput )
{
   while( true )
   {
      // ------------------------------------------------3
      // Read image from images list

      // - Read list line

      string imageName;
      getline( siParams.listFile, imageName );

      if( siParams.listFile.eof() || listLine.empty() )
      {
         break;
      }

      // - Read image

      Mat inputImage = imread( imageName.c_str() );

      if( inputImage.empty() )
      {
         cerr << "  Unable to load image: " << imageName << endl
              << endl;
         return 1;
      }
```

```
        // ---------------------------------------------------3
        // Detect traffic lights bounding boxes using cascade classifier

        vector<Rect> boundingBoxes;

        trafficLightsDetection( inputImage, boundingBoxes, tldParams );

        // ---------------------------------------------------3
        // Detect spot lights bounding blobs using blob detector

        vector<Rect> boundingBlobs;

        spotLightsDetection( inputImage, boundingBoxes, boundingBlobs, sldParams );

        // ---------------------------------------------------3
        // Recognize spot lights color by position, matching or hue

        vector<slColor> spotLightColors;

        spotLigthsRecognition( inputImage, boundingBoxes, boundingBlobs, spotLightColors,
                               slrParams );

        // ---------------------------------------------------3
        // Display traffic lights detection and recognition

        detectionAndRecognitionDisplay( inputImage, boundingBoxes, boundingBlobs,
                                        spotLightColors, drdParams );

        // ---------------------------------------------------3
        // Wait for key press to exit or continue

        char keyPress = waitKey( 1 );

        if( keyPress == 27 )
        {
            break;
        }
    }
}

// ---------------------------------------------------------------2
// Traffic lights detection and recognition from video file

if( siParams.videoInput )
{
    int frameNum = 0;

    while( true )
    {
        // ---------------------------------------------------3
        // Grab video frame

        Mat videoFrame;
        siParams.videoCapture >> videoFrame;

        if( videoFrame.empty() )
        {
            cerr << "  Unable to grab video frame: " << frameNum << endl
                 << endl;

            return 1;
        }

        // ---------------------------------------------------3
        // Detect and recognize traffic lights

        vector<Rect> boundingBoxes;
        vector<Rect> boundingBlobs;
        vector<slColor> spotLightColors;

        if( !( frameNum % siParams.frameRate ) )
        {
```

```
            // ------------------------------------4
            // Detect traffic lights bounding boxes using cascade classifier

            trafficLightsDetection( videoFrame, boundingBoxes, tldParams );

            // ------------------------------------4
            // Detect spot lights bounding blobs using blob detector

            spotLightsDetection( videoFrame, boundingBoxes, boundingBlobs, sldParams );

            // ------------------------------------4
            // Recognize spot lights color by position, matching or hue

            spotLigthsRecognition( videoFrame, boundingBoxes, boundingBlobs,
                                   spotLightColors, slrParams );
        }

        // ------------------------------------------------3
        // Display traffic lights detection and recognition

        detectionAndRecognitionDisplay( videoFrame, boundingBoxes, boundingBlobs,
                                        spotLightColors, drdParams );

        // ------------------------------------------------3
        // Wait for key press to exit or continue

        char keyPress = waitKey( 1 );

        if( keyPress == 27 )
        {
            break;
        }

    }
}

// ---------------------------------------------------------------2
// Traffic lights detection and recognition from video camera

if( siParams.cameraInput )
{
    int frameNum = 0;

    while( true )
    {
        // ------------------------------------------------3
        // Grab camera frame

        Mat cameraFrame;
        siParams.videoCapture >> cameraFrame;

        if( cameraFrame.empty() )
        {
            cerr << "  Unable to grab video frame: " << frameNum << endl
                 << endl;

            return 1;
        }

        // ------------------------------------------------3
        // Detect and recognize traffic lights

        vector<Rect> boundingBoxes;
        vector<Rect> boundingBlobs;
        vector<slColor> spotLightColors;

        if( !( frameNum % siParams.frameRate ) )
        {
            // ------------------------------------4
            // Detect traffic lights bounding boxes using cascade classifier

            trafficLightsDetection( cameraFrame, boundingBoxes, tldParams );
```

```
            // ------------------------------------4
            // Detect spot lights bounding blobs using blob detector

            spotLightsDetection( cameraFrame, boundingBoxes, boundingBlobs, sldParams );

            // ------------------------------------4
            // Recognize spot lights color by position, matching or hue

            spotLigthsRecognition( cameraFrame, boundingBoxes, boundingBlobs,
                                   spotLightColors, slrParams );
         }

         // -------------------------------------------------3
         // Display traffic lights detection and recognition

         detectionAndRecognitionDisplay( cameraFrame, boundingBoxes, boundingBlobs,
                                         spotLightColors, drdParams );

         // -------------------------------------------------3
         // Wait for key press to exit or continue
         char keyPress = waitKey( 1 );

         if( keyPress == 27 )
         {
            break;
         }

      }
   }

   cout << "  Exit" << endl
        << endl;

   return 0;

}
```

5.9. *script_tldr.sh*

```
#!/bin/bash

#Establecer la ruta de la aplicacion "tldr"
application="build/tldr"

#Configurar los parametros de la aplicación                                    #<default>

# - Fuente de entrada de imagenes
si_list_file_name="Files/Source_Input/list.txt"                               #<null>
si_video_file_name=""                                                         #<null>
si_camera_number=-1                                                           #<-1>
si_frame_rate=5                                                               #<5>

# - Deteccion de los semaforos
tld_cascade_file_name="Files/Classifier/cascade.xml"                          #<null>
tld_preprocess_shrink_factor=0.5                                              #<0.5>
tld_preprocess_crop_factor=0.5                                                #<0.5>
tld_preprocess_blur_kernel_size=3                                             #<3>
tld_detection_scale_factor=1.1                                                #<1.1>
tld_detection_min_width=0                                                     #<0>
tld_detection_min_height=0                                                    #<0>
tld_detection_max_width=0                                                     #<0>
tld_detection_max_height=0                                                    #<0>
tld_min_bounding_boxes_group=1                                                #<1>
tld_min_bounding_boxes_overlap=0.0                                            #<0.5>
tld_max_bounding_boxes_number=3                                               #<1>

# - Deteccion de la luz de los semaforos
sld_preprocess_blur_kernel_size=3                                             #<3>
sld_detection_threshold_step=5                                                #<5>
sld_detection_min_threshold=0                                                 #<0>
sld_detection_max_threshold=254                                               #<254>
sld_detection_min_diameter=10                                                 #<10>
sld_detection_max_diameter=40                                                 #<40>
sld_detection_min_circularity=0.8                                             #<0.8>
sld_detection_max_circularity=1.0                                             #<1.0>
sld_detection_min_inertia_ratio=0.8                                           #<0.8>
sld_detection_max_inertia_ratio=1.2                                           #<1.2>
sld_detection_min_convexity=0.8                                               #<0.8>
sld_detection_max_convexity=1.2                                               #<1.2>
sld_min_bounding_blobs_group=1                                                #<1>
sld_min_bounding_blobs_overlap=0.0                                            #<0.5>

# - Deteccion de la luz de los semaforos
slr_recognition_type="pos-hue" #<{ pos | match | hue  | pos-hue | match-hue | null (*) }>

# - Reconocimiento de la luz de los semaforos por posicion
slrp_blur_kernel_size=3                                                       #<3>
slrp_sobel_kernel_size=3                                                      #<3>
slrp_sobel_edges_threshold=100                                                #<100>
slrp_bound_lines_of_lights_factor=0.25                                        #<0.25>
slrp_edges_width_factor=0.8                                                   #<0.8>
slrp_distance_between_lights_factor=1.5 slrm_green_light_box_y_location=090   #<-1>
slrm_green_light_box_width_size=35                                            #<-1>
slrm_green_light_box_height_size=35                                           #<-1>

# - Reconocimiento de la luz de los semaforos por tono
slrh_blur_kernel_size=3                                                       #<3>
slrh_internal_halo_factor=0.25                                                #<0.25>
slrh_external_halo_factor=1.25                                                #<1.25>
slrh_filter_histogram_factor=0.5                                              #<0.5>
slrh_max_red_hue=150                                                          #<130>
slrh_min_red_hue=125                                                          #<120>
slrh_max_amber_hue=120                                                        #<100>
slrh_min_amber_hue=90                                                         #<90>
slrh_max_green_hue=40                                                         #<40>
slrh_min_green_hue=20                                                         #<30>
```

```
# - Visualizacion de la deteccion y el reconocimiento de semaforos
drd_red_image_file_name="Files/Recognition_Images/red.jpg"                          #<null>
drd_amber_up_image_file_name="Files/Recognition_Images/amberup.jpg"                 #<null>
drd_green_image_file_name="Files/Recognition_Images/amberdown.jpg"                  #<null>
drd_amber_down_image_file_name="Files/Recognition_Images/green.jpg"                 #<null>
drd_amber_or_green_image_file_name="Files/Recognition_Images/amberdown.jpg"         #<null>
drd_amber_up_or_down_image_file_name"Files/Recognition_Images/amberorgreen.jpg"     #<null>
drd_unknown_image_file_name="Files/Recognition_Images/unknown.jpg"                  #<null>
drd_detection_draw_color="blue"                                                     #<blue>
drd_detection_draw_thickness=4                                                      #<4>
drd_display_scale_factor=0.5                                                        #<0.5>

# - Ayuda de uso
help="false"                                                                        #<false>
                                                                                    #<1.5>


# - Reconocimiento de la luz de los semaforos por matching
slrm_red_template_file_name="Files/Matching_Templates/red.jpg"                      #<null>
slrm_amber_template_file_name="Files/Matching_Templates/amber.jpg"                  #<null>
slrm_green_template_file_name="Files/Matching_Templates/green.jpg"                  #<null>
slrm_red_light_box_x_location=0                                                     #<-1>
slrm_red_light_box_y_location=0                                                     #<-1>
slrm_red_light_box_width_size=35                                                    #<-1>
slrm_red_light_box_height_size=35                                                   #<-1>
slrm_amber_light_box_x_location=0                                                   #<-1>
slrm_amber_light_box_y_location=45                                                  #<-1>
slrm_amber_light_box_width_size=35                                                  #<-1>
slrm_amber_light_box_height_size=35                                                 #<-1>
slrm_green_light_box_x_location=0                                                   #<-1>


# Establecer los parametros de la aplicacion
parameters=""

# - Fuente de entrada de imagenes
if [ "$si_list_file_name" != "null" ]; then
   parameters="$parameters -siListFileName $si_list_file_name"
fi
if [ "$si_video_file_name" != "null" ]; then
   parameters="$parameters -siVideoFileName $si_video_file_name"
fi
if [ "$si_camera_number" != "-1" ]; then
   parameters="$parameters -siCameraNumber $si_camera_number"
fi
if [ "$si_frame_rate" != "-1" ]; then
   parameters="$parameters -siFrameRate $si_frame_rate"
fi

# - Deteccion de los semaforos
if [ "$tld_cascade_file_name" != "null" ]; then
   parameters="$parameters -tldCascadeFileName $tld_cascade_file_name"
fi
if [ "$tld_preprocess_shrink_factor" != "-1" ]; then
   parameters="$parameters -tldShrinkFactor $tld_preprocess_shrink_factor"
fi
if [ "$tld_preprocess_crop_factor" != "-1" ]; then
   parameters="$parameters -tldCropFactor $tld_preprocess_crop_factor"
fi
if [ "$tld_preprocess_blur_kernel_size" != "-1" ]; then
   parameters="$parameters -tldBlurSize $tld_preprocess_blur_kernel_size"
fi
if [ "$tld_detection_scale_factor" != "-1" ]; then
   parameters="$parameters -tldScaleFactor $tld_detection_scale_factor"
fi
if [ "$tld_detection_min_width" != "-1" ]; then
   parameters="$parameters -tldMinWidth $tld_detection_min_width"
fi
if [ "$tld_detection_min_height"  != "-1" ]; then
   parameters="$parameters -tldMinHeight $tld_detection_min_height"
fi
if [ "$tld_detection_max_width" != "-1" ]; then
   parameters="$parameters -tldMaxWidth $tld_detection_max_width"
fi
```

```
if [ "$tld_detection_max_height" != "-1" ]; then
    parameters="$parameters -tldMaxHeight $tld_detection_max_height"
fi
if [ "$tld_min_bounding_boxes_group" != "-1" ]; then
    parameters="$parameters -tldMinGroup $tld_min_bounding_boxes_group"
fi
if [ "$tld_min_bounding_boxes_overlap" != "-1" ]; then
    parameters="$parameters -tldMinOverlap $tld_min_bounding_boxes_overlap"
fi
if [ "$tld_max_bounding_boxes_number" != "-1" ]; then
    parameters="$parameters -tldMaxNumber $tld_max_bounding_boxes_number"
fi

# - Deteccion de la luz de los semaforos
if [ "$sld_preprocess_blur_kernel_size" != "-1" ]; then
    parameters="$parameters -sldBlurSize $sld_preprocess_blur_kernel_size"
fi
if [ "$sld_detection_threshold_step" != "-1" ]; then
    parameters="$parameters -sldThreshStep $sld_detection_threshold_step"
fi
if [ "$sld_detection_min_threshold" != "-1" ]; then
    parameters="$parameters -sldMinThresh $sld_detection_min_threshold"
fi
if [ "$sld_detection_min_threshold" != "-1" ]; then
    parameters="$parameters -sldMinThresh $sld_detection_min_threshold"
fi
if [ "$sld_detection_max_threshold" != "-1" ]; then
    parameters="$parameters -sldMaxThresh $sld_detection_max_threshold"
fi
if [ "$sld_detection_min_diameter" != "-1" ]; then
    parameters="$parameters -sldMinDiam $sld_detection_min_diameter"
fi
if [ "$sld_detection_max_diameter" != "-1" ]; then
    parameters="$parameters -sldMaxDiam $sld_detection_max_diameter"
fi
if [ "$sld_detection_min_circularity" != "-1" ]; then
    parameters="$parameters -sldMinCirc $sld_detection_min_circularity"
fi
if [ "$sld_detection_max_circularity" != "-1" ]; then
    parameters="$parameters -sldMaxCirc $sld_detection_max_circularity"
fi
if [ "$sld_detection_min_inertia_ratio" != "-1" ]; then
    parameters="$parameters -sldMinInert $sld_detection_min_inertia_ratio"
fi
if [ "$sld_detection_max_inertia_ratio" != "-1" ]; then
    parameters="$parameters -sldMaxInert $sld_detection_max_inertia_ratio"
fi
if [ "$sld_detection_min_convexity" != "-1" ]; then
    parameters="$parameters -sldMinConvex $sld_detection_min_convexity"
fi
if [ "$sld_detection_max_convexity" != "-1" ]; then
    parameters="$parameters -sldMaxConvex $sld_detection_max_convexity"
fi
if [ "$sld_min_bounding_blobs_group" != "-1" ]; then
    parameters="$parameters -sldMinGroup $sld_min_bounding_blobs_group"
fi
if [ "$sld_min_bounding_blobs_overlap" != "-1" ]; then
    parameters="$parameters -sldMinOverlap $sld_min_bounding_blobs_overlap"
fi

# - Reconocimiento de la luz de los semaforos
if [ "$slr_recognition_type" != "null" ]; then
    parameters="$parameters -slrRecognType $slr_recognition_type"
fi

# - Reconocimiento de la luz de los semaforos por posicion
if [ "$slrp_blur_kernel_size" != "-1" ]; then
    parameters="$parameters -slrpBlurSize $slrp_blur_kernel_size"
fi
if [ "$slrp_sobel_kernel_size"  != "-1" ]; then
    parameters="$parameters -slrpSobelSize $slrp_sobel_kernel_size"
fi
if [ "$slrp_sobel_edges_threshold" != "-1" ]; then
```

```
    parameters="$parameters -slrpEdgesThresh $slrp_sobel_edges_threshold"
fi
if [ "$slrp_bound_lines_of_lights_factor" != "-1" ]; then
    parameters="$parameters -slrpBoundFactor $slrp_bound_lines_of_lights_factor"
fi
if [ "$slrp_edges_width_factor" != "-1" ]; then
    parameters="$parameters -slrpWidthFactor $slrp_edges_width_factor"
fi
if [ "$slrp_distance_between_lights_factor" != "-1" ]; then
    parameters="$parameters -slrpDistFactor $slrp_distance_between_lights_factor"
fi

# - Reconocimiento de la luz de los semaforos por matching
if [ "$slrm_red_template_file_name" != "null" ]; then
    parameters="$parameters -slrmRedTemplFileName $slrm_red_template_file_name"
fi
if [ "$slrm_amber_template_file_name" != "null" ]; then
    parameters="$parameters -slrmAmberTemplFileName $slrm_amber_template_file_name"
fi
if [ "$slrm_green_template_file_name" != "null" ]; then
    parameters="$parameters -slrmGreenTemplFileName $slrm_green_template_file_name"
fi
if [ "$slrm_red_light_box_x_location" != "-1" ]; then
    parameters="$parameters -slrmRedLightBoxX $slrm_red_light_box_x_location"
fi
if [ "$slrm_red_light_box_y_location" != "-1" ]; then
    parameters="$parameters -slrmRedLightBoxY $slrm_red_light_box_y_location"
fi
if [ "$slrm_red_light_box_width_size" != "-1" ]; then
    parameters="$parameters -slrmRedLightBoxW $slrm_red_light_box_width_size"
fi
if [ "$slrm_red_light_box_height_size" != "-1" ]; then
    parameters="$parameters -slrmRedLightBoxH $slrm_red_light_box_height_size"
fi
if [ "$slrm_amber_light_box_x_location" != "-1" ]; then
    parameters="$parameters -slrmAmberLightBoxX $slrm_amber_light_box_x_location"
fi
if [ "$slrm_amber_light_box_x_location" != "-1" ]; then
    parameters="$parameters -slrmAmberLightBoxX $slrm_amber_light_box_x_location"
fi
if [ "$slrm_amber_light_box_y_location" != "-1" ]; then
    parameters="$parameters -slrmAmberLightBoxY $slrm_amber_light_box_y_location"
fi
if [ "$slrm_amber_light_box_width_size" != "-1" ]; then
    parameters="$parameters -slrmAmberLightBoxW $slrm_amber_light_box_width_size"
fi
if [ "$slrm_amber_light_box_height_size" != "-1" ]; then
    parameters="$parameters -slrmAmberLightBoxH $slrm_amber_light_box_height_size"
fi
if [ "$slrm_green_light_box_x_location" != "-1" ]; then
    parameters="$parameters -slrmGreenLightBoxX $slrm_green_light_box_x_location"
fi
if [ "$slrm_green_light_box_y_location" != "-1" ]; then
    parameters="$parameters -slrmGreenLightBoxY $slrm_green_light_box_y_location"
fi
if [ "$slrm_green_light_box_width_size" != "-1" ]; then
    parameters="$parameters -slrmGreenLightBoxW $slrm_green_light_box_width_size"
fi
if [ "$slrm_green_light_box_height_size" != "-1" ]; then
    parameters="$parameters -slrmGreenLightBoxH $slrm_green_light_box_height_size"
fi
if [ "$slrm_blur_kernel_size" != "-1" ]; then
    parameters="$parameters -slrmBlurSize $slrm_blur_kernel_size"
fi

# - Reconocimiento de la luz de los semaforos por tono
if [ "$slrh_blur_kernel_size" != "-1" ]; then
    parameters="$parameters -slrhBlurSize $slrh_blur_kernel_size"
```

```
fi
if [ "$internal_halo_factor" != "-1" ]; then
   parameters="$parameters -slrhIntHaloFactor $internal_halo_factor"
fi
if [ "$external_halo_factor" != "-1" ]; then
   parameters="$parameters -slrhExtHaloFactor $external_halo_factor"
fi
if [ "$slrh_filter_histogram_factor" != "-1" ]; then
   parameters="$parameters -slrhHistFactor $slrh_filter_histogram_factor"
fi
if [ "$slrh_max_red_hue" != "-1" ]; then
   parameters="$parameters -slrhMaxRedHue $slrh_max_red_hue"
fi
if [ "$slrh_min_red_hue" != "-1" ]; then
   parameters="$parameters  -slrhMinRedHue $slrh_min_red_hue"
fi
if [ "$slrh_max_amber_hue"  != "-1" ]; then
   parameters="$parameters -slrhMaxAmberHue $slrh_max_amber_hue"
fi
if [ "$slrh_min_amber_hue" != "-1" ]; then
   parameters="$parameters -slrhMinAmberHue $slrh_min_amber_hue"
fi
if [ "$slrh_max_green_hue" != "-1" ]; then
   parameters="$parameters -slrhMaxGreenHue $slrh_max_green_hue"
fi
if [ "$slrh_min_green_hue" != "-1" ]; then
   parameters="$parameters -slrhMinGreenHue $slrh_min_green_hue"
fi

# - Visualizacion de la deteccion y el reconocimiento
if [ "$drd_red_image_file_name" != "null" ]; then
   parameters="$parameters -drdRedImageFileName $drd_red_image_file_name"
fi
if [ "$drd_amber_up_image_file_name" != "null" ]; then
   parameters="$parameters -drdAmberUpImageFileName $drd_amber_up_image_file_name"
fi
if [ "$drd_green_image_file_name" != "null" ]; then
   parameters="$parameters -drdGreenImageFileName $drd_green_image_file_name"
fi
if [ "$drd_amber_down_image_file_name" != "null" ]; then
   parameters="$parameters -drdAmberDownImageFileName $drd_amber_down_image_file_name"
fi
if [ "$drd_amber_or_green_image_file_name" != "null" ]; then
   parameters="$parameters -drdAmberOrGreenImageFileName
               $drd_amber_or_green_image_file_name"
fi
if [ "$drd_amber_up_or_down_image_file_name" != "null" ]; then
   parameters="$parameters -drdAmberUpOrDownImageFileName
               $drd_amber_up_or_down_image_file_name"
fi
if [ "$drd_unknown_image_file_name" != "null" ]; then
   parameters="$parameters -drdUnknownImageFileName $drd_unknown_image_file_name"
fi
if [ "$drd_detection_draw_color" != "-1" ]; then
   parameters="$parameters -drdDrawColor $drd_detection_draw_color"
fi
if [ "$drd_detection_draw_thickness" != "-1" ]; then
   parameters="$parameters -drdDrawThickness $drd_detection_draw_thickness"
fi
if [ "$drd_display_scale_factor" != "-1" ]; then
   parameters="$parameters -drdScaleFactor $drd_display_scale_factor"
fi
# - Help usage
if [ "$help_usage" == "true" ]; then
```

```
    parameters="$parameters -help"
fi

#Ejecutar la aplicacion
$application $parameters

#Pausa
echo "+------------------------------------------------------------------------------+"
echo "Pulse enter para salir"
echo ""
read -n 0 -ers

#Salir
exit
```

Capítulo 6

Presupuesto

6.1. Introducción

En este capítulo se presentan los datos económicos referentes a los costes que implica la realización del presente proyecto. En el presupuesto se incluyen:

- Costes de personal.
- Costes por uso de equipos y material.
- Costes totales
- Costes de ejecución por contrata.
- Presupuesto total.

Para el cálculo de costes, se tiene que considerar que la duración del proyecto es de **12 meses**.

6.2. Costes de Personal

Concepto	**Trabajo Realizado**	**Coste/Mes**	**Dedicación**	**Coste**
Ingeniero Industrial	Estudio Investigación Diseño Programación Pruebas Redacción	1.953€	100%	23.436€
TOTAL				**23.436€**

6.3. Costes por Uso de Equipos y Material

Concepto	**Descripción**	**Coste**	**Amortización**	**Uso**	**Coste/Uso**
Ordenador Personal de Trabajo	Compaq Evo D510 Windows XP	1.035€	120 meses	12 meses	104€
Equipo de Entrenamiento	AZserver MN + Pro Ubuntu 12.04 LTS	2.712€	60 meses	6 meses	271€
Software de Ofimática	Microsoft Office 2007	491€	72 meses	12 meses	82€
Software de Programación	Microsoft Visual Studio 2010	0€	-	-	0€
Librerías de Programación	OpenCV versión 2.4.6	0€	-	-	0€
Cámara de Grabación	Iphone 4 de 32 GB	677€	60 meses	1 mes	11€
Soporte de Cámara para Vehículos	Soporte Iphone 4 para Vehículos	15€	60 meses	1 mes	1€
Vehículo de Grabación	Opel Astra del año 1997	10.431€	192 meses	1 mes	54€
TOTAL					**523€**

6.4. Costes Totales

Concepto	**Porcentaje**	**Coste**
Costes Personal	-	23.436€
Costes por Uso de Equipos y Material	-	523€
Costes Indirectos	20%	4.792€
TOTAL		**28.751€**

6.5. Costes de Ejecución por Contrata

Concepto	**Porcentaje**	**Coste**
Coste Total	-	28.751€
Beneficio Industrial	6%	1.725€
Honorarios de Dirección y Redacción	10%	2.875€
TOTAL		**33.351€**

6.6. Presupuesto Total

Concepto	**Porcentaje**	**Coste**
Costes de Ejecución por Contrata	-	33.351€
I.V.A.	21%	7.004€
TOTAL		**40.355€**

El presupuesto total del proyecto asciende a *CUARENTA MIL TRESCIENTOS CINCUENTA Y CINCO EUROS.*

Leganés, a 31 de Octubre de 2013

Fdo: Víctor Alonso Mendieta.

Capítulo 7

Referencias

[1] http://en.wikipedia.org/wiki/Advanced_driver_assistance_systems, (2013).

[2] C. Papageorgiou, M. Oren and T. Poggio. “A General Framework for Object Detection”. International Conference on Computer Vision, (1998).

[3] P. Viola and M.J. Jones. “Robust Real-Time Face Detection”. International Journal of Computer Vision, vol. 57, pp. 137-154, (2004).

[4] R. Lienhart and J. Maydt. “An Extended Set of Haar-like Features for Rapid Object Detection”. IEEE ICIP 2002, vol. 1, pp. 900-903, (2002).

[5] DC. He and L. Wang. “Texture Unit, Texture Spectrum, and Texture Analysis”. IEEE Trans. Geoscience and Remote Sensing, vol. 28, pp. 509-512, (1990).

[6] T. Ojala, M. Pietikäinen, and D. Harwood. “Performance Evaluation of Texture Measures with Classification Based on Kullback Discrimination of Distributions”. IEEE ICPR, vol. 1, pp. 582-585, (1994).

[7] T. Ojala, M. Pietikäinen, and T. Mäenpää. “Multiresolution Gray-scale and Rotation Invariant Texture Classification with Local Binary Patterns”. IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 24, pp. 971-987, (2002).

[8] R. Lienhart, A. Kuranov, and V. Pisarevsky. “Empirical Analysis of Detection Cascades of Boosted Classifiers for Rapid Object Detection”. Intel Labs, MRL Technical Report, (2002).

[9] R.E. Schapire. “The Strength of Weak Learnability”. 30th Annual Symposium on Foundations of Computer Science, pp. 28-33, (1989).

[10] Y. Freund and R.E. Schapire. “Experiments with a New Boosting Algorithm”. Proceedings of the Thirteenth International Conference on Machine Learning, (1996).

[11] Y. Freund and R.E. Schapire. “A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting”. Journal of Computer and System Sciences, vol. 55, pp. 119-139, (1996).

[12] R.E. Schapire and Y. Singer. “Improved Boosting Algorithms Using Confidence-Rated Predictions”. Proceedings of the Eleventh Annual Conference on Computational Learning Theory, pp. 80-91, (1990).

[13] J. Friedman, T. Hastie and R. Tibshirani. “Additive Logistic Regression: a Statistical View of Boosting”. The Annals of Statistics, vol. 28, pp. 337-374, (1998).

[14] T. Fawcett. “ROC Graphs: Notes and Practical Considerations for Researchers”. HP Laboratories Technical Report, (2004).

[15] C. D. Manning, P. Raghavan, and H. Schutze. “An Introduction to Information Retrieval”. Cambridge University Press, (2008).

[16] A. Martin, G. Doddington, T. Kamm, M. Ordowski, y M. Przybocki. “The DET curve in assessment of detection task performance”. Proceedings Conference on Speech Communication and Technology, vol. 4, pp. 1899-1903, (1997).

[17] http://en.wikipedia.org/wiki/Receiver_operating_characteristic, (2013).

[18] http://en.wikipedia.org/wiki/Detection_error_tradeoff, (2013).

[19] http://en.wikipedia.org/wiki/Precision_and_recall, (2013).

[20] http://www.vlfeat.org/overview/plots-rank.html, (2013).

[21] http://www.vlfeat.org/matlab/vl_roc.html, (2013).

[22] http://www.vlfeat.org/matlab/vl_det.html, (2013).

[23] http://www.vlfeat.org/matlab/vl_pr.html, (2013).

[24] https://github.com/vlfeat/vlfeat/blob/master/toolbox/plotop/vl_det.m, (2013).

[25] https://github.com/vlfeat/vlfeat/blob/master/toolbox/plotop/vl_pr.m, (2013).

[26] https://github.com/vlfeat/vlfeat/blob/master/toolbox/plotop/vl_roc.m, (2013).

[27] http://en.wikipedia.org/wiki/IPhone_4, (2013).

[28] http://en.wikipedia.org/wiki/List_of_Ubuntu_releases, (2013).

[29] http://ark.intel.com/products/64594, (2013)

[30] http://www.nvidia.com/docs/IO/43395/NV-DS-Tesla-C2075.pdf, (2013)

[31] http://docs.opencv.org/index.html, (2013).

Nada es Imposible